A3ME – Device-Agent based Middleware for Mixed Mode Environments

Geräteagentenbasierte Middleware für heterogene Netzwerke und Umgebungen
Zur Erlangung des akademischen Grades Doktor-Ingenieur (Dr.-Ing.)
genehmigte Dissertation von Dipl.-Inf. Arthur Herzog aus Romanowka
Tag der Einreichung: 23. Juli 2015, Tag der Prüfung: 3. November 2015
Darmstadt 2016 — D 17

1. Gutachten: Prof. Alejandro P. Buchmann, PhD
2. Gutachten: Prof. Dr.-Ing. Matthias Hollick

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Fachbereich Informatik
Datenbanken und Verteilte Systeme

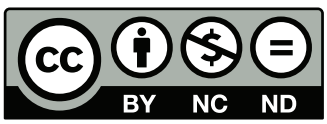
A3ME – Device-Agent based Middleware for Mixed Mode Environments
Geräteagentenbasierte Middleware für heterogene Netzwerke und Umgebungen

Genehmigte Dissertation von Dipl.-Inf. Arthur Herzog aus Romanowka

1. Gutachten: Prof. Alejandro P. Buchmann, PhD
2. Gutachten: Prof. Dr.-Ing. Matthias Hollick

Tag der Einreichung: 23. Juli 2015
Tag der Prüfung: 3. November 2015

Darmstadt 2016 — D 17



To my family.

Abstract

The Internet of Things describes a vision and a process, that partially already takes place, in which the things become interactive by being provided with minimal computing power and communication capability to support the people in their tasks. These smart things or devices are nowadays prevailingly organized in island solutions and are often not able to interact across the border of their own island.

As the amount of electronic devices we are surrounded by increases every day not only in numbers but also by their variety, enabling interoperability among these devices became crucial. Mixed Mode Environments (MME) refer to networks composed of very different kinds of devices, which are distributed among various physical environments and communicate with each other using different communication technologies. The single nodes in the network can be sensors, actuators, robots, unmanned vehicles (UV), computers, user interfaces, smartphones, etc. All those devices have their specific capabilities and constraints. Many of these devices are manufactured by different companies and use different software and operating systems. Those devices can communicate with each other by wire, radio, light, sound or other transmission medium. For each of these transmission media many different communication technologies exist, which use different protocols, frequencies, encodings, etc.

Nowadays application developers have to deal with the before mentioned heterogeneity when developing a new application for such a network. Each time when a new kind of node appears the application has to be adjusted to deal with the new hardware. Middleware is a way to avoid this direct interaction of applications with the different hardware and software of the devices. So middleware has to abstract over all the different devices, their capabilities and communication technologies and to offer the applications uniform interfaces to interact with those.

The Device-Agent based Middleware for Mixed Mode Environments (A3ME) framework developed in this dissertation enables interoperability among different nodes without the need of adjustments each time new hardware is introduced. Our approach offers an abstraction for the different hardware: it sees all the different nodes in the network as independent entities, we call them device-agents. These device-agents know the capabilities and constraints of the respectively represented device and represent those in a neutral format developed in this dissertation. This neutral representation is independent of the technologies used on the represented device. Depending on the capabilities of the represented device a device-agent offers services to other agents and can also use services of other device-agents. The complexity of device-agent running for example on a small sensor node and on an UV can vary considerably. Thus a sensor-agent might be only capable to measure the current temperature and to send it to someone who is interested in this data, whereas the UV-agent can move through the environment, collect the data from the sensor-agents, aggregate, evaluate the collected data and use the gathered information further in its decision making process.

Communication between different nodes is defined technology independently and can be applied to different communication technologies. Descriptions of the devices and their capabilities are based on a common classification developed in this work. This allows interactions based on capability-classes. All interactions are message based and each message belongs to a specific performative, which corresponds to the type of action the message represents. The structure of the messages is defined using Abstract Syntax Notation One (ASN.1) allowing to define their content dynamically. The use of ASN.1 Packed Encoding Rules allows to encode the messages in a very byte-length efficient way.

This framework offers the basis to enable interoperability between heterogeneous devices directly and among various island solutions for electronic devices, which exist in different areas like multimedia, personal communication, smart home, smart office, connected car, etc.

Zusammenfassung

Das Internet der Dinge ist eine Vision und zum Teil bereits stattfindende Entwicklung, in der die (Alltags-)Gegenstände durch Ausstattung mit geringer Rechenleistung und Kommunikationsfähigkeiten interaktiv werden und den Menschen bei seinen Tätigkeiten unterstützen. Diese Geräte sind heutzutage überwiegend als Insellösungen organisiert und haben in vielen Fällen keine inselübergreifende Interaktionsmöglichkeit.

Die Anzahl von solchen Geräten wächst kontinuierlich – nicht nur in deren Anzahl sondern auch in deren Vielfalt. Deshalb ist es unabdingbar eine Möglichkeit zu schaffen, mit der diese Geräte miteinander bei Bedarf interagieren können. Bereits heutzutage sind wir von sehr verschiedenartigen Geräten in unserer Umwelt umgeben. Zum Teil sind diese Geräte vernetzt und kommunizieren untereinander. Es handelt sich dabei zum Beispiel um die persönlichen Kommunikationsgeräte, Multimediageräte, Sensoren, Computer usw. All diese Geräte haben ihre spezifischen Fähigkeiten und Einschränkungen. Meistens wurden diese von unterschiedlichen Herstellern produziert, mit Hilfe verschiedener Programmiersprachen implementiert, und verwenden unterschiedliche Betriebssysteme. Die Geräte können über Draht, Funk, Licht, Schall oder auch über andere Medien kommunizieren. Für jedes dieser Kommunikationsmedien existieren mehrere verschiedene Kommunikationstechnologien, die unterschiedliche Protokolle, Frequenzen, Kodierungen usw. verwenden.

Häufig sind die Entwickler mit der gerade beschriebenen Heterogenität konfrontiert, wenn sie eine neue Anwendung für so ein Netzwerk entwickeln. Jedes mal, wenn ein neuartiges Gerät erscheint, muss die Anwendung angepasst werden, um mit dem neuen Gerät umgehen zu können. Die Verwendung einer Middleware ermöglicht es diese direkte Interaktion der Anwendungen mit den unterschiedlichen Geräten zu vermeiden. Somit muss die Middleware die Eigenschaften, Einschränkungen, und Fähigkeiten der Geräte abstrahieren und den Anwendungen eine vereinheitlichte Schnittstelle für die Verwendung dieser anbieten.

Die in dieser Dissertation entwickelte geräteagentenbasierte Middleware für heterogene Netzwerke und Umgebungen (A3ME) ermöglicht Interaktionen zwischen unterschiedlichen elektronischen Geräten ohne, dass für jede neu hinzukommende Geräteart Anpassungen vorgenommen werden müssen. Unsere Lösung bietet eine Abstraktion für die verschiedenen elektronischen Geräte: Sie betrachtet alle Geräte als unabhängige Entitäten – Geräteagenten. Diese Geräteagenten kennen die Fähigkeiten und Einschränkungen des jeweils repräsentierten Gerätes und stellen diese in einer neutralen in dieser Arbeit entwickelten Darstellung dar. Diese neutrale Darstellung ist unabhängig von der auf dem jeweiligen Gerät verwendeten Technologie. Abhängig von den Fähigkeiten der repräsentierten Geräte bieten die Geräteagenten Dienste anderen Geräteagenten an und können bei Bedarf Dienste anderer Geräteagenten verwenden.

Die Kommunikation zwischen den Geräten ist definiert unabhängig von den spezifischen Kommunikationstechnologien und kann auf unterschiedliche Kommunikationstechniken abgebildet werden. Beschreibungen von Geräten und deren Fähigkeiten basieren auf einer gemeinsamen Klassifikation. Dies ermöglicht die Interaktionen basierend auf Typen von Fähigkeiten. Alle Interaktionen sind nachrichtenbasiert und jede Nachricht ist einem Nachrichtentyp zugeordnet, entsprechend dem Typ der beabsichtigten Aktion. Alle hier benutzten Nachrichten sind in ASN.1 (Abstract Syntax Notation One) definiert und ermöglichen den Inhalt dynamisch aufzubauen. In Kombination mit ASN.1 Packed encoding Rules (PER) können diese Nachrichten sehr effizient in Bezug auf die Nachrichtengröße kodiert werden.

Die in dieser Dissertation beschriebene Lösung bietet die Basis für die Interaktionen zwischen den unterschiedlichen Geräten direkt und zwischen den verschiedenen existierenden Insellösungen wie Multimedia, persönliche Kommunikation, intelligentes Zuhause, intelligentes Büro, vernetztes Auto usw.

Contents

List of Tables

List of Figures

1 Introduction

This work describes the development of a new middleware to enable interactions of heterogeneous devices in Mixed Mode Environments (MME). The Mixed Mode Environments refer to networks composed of devices with different dimensions of heterogeneity. These devices are distributed among various physical environments and communicate with each other using different communication technologies. Individual nodes in the network can be sensors, actuators, mobile phones, multimedia devices, robots, large servers, etc. All these devices have their specific capabilities and constraints, are manufactured by different companies and use different software and operating systems. Some nodes do not even have an operating system. The physical environment can also be heterogeneous like indoors, outdoors, underground, underwater, etc. These devices can communicate with each other in different ways: by wire, radio, infrared, light, sound or through other media. For each of these communication media, many different communication technologies exist, which use different protocols, frequencies, encoding schemata, etc.

In MME different research areas come together: Wireless Sensor Networks (WSNs), Wireless Sensor and Actor Networks (WSANs), Ubiquitous Computing, Robotics, etc. Most research in the WSN area is done in homogeneous networks using the same kind of device for all nodes in the network or in combination with one more powerful device that is used as a gateway and data sink. In WSANs an additional type of nodes – actors – is used. While sensors are only observing the physical world, actors are capable of interacting with the physical world. In Ubiquitous Computing multiple devices in people's surroundings are performing tasks without people necessarily interacting with the devices. Here a broad variety of devices are involved like media devices, mobile phones, light and temperature control devices, etc. Robots usually have both sensors and actuators. The robots themselves also can be seen as actors. They can interact with humans, with other devices and with the environment. [92]

Today, application developers must deal with this heterogeneity when developing a new application for heterogeneous networks. Each time a new kind of node appears in the network, the applications have to be adjusted to deal with the new hardware. Middleware is a way to avoid this direct interaction of applications with the hardware and software of the devices, and to enable and to simplify the interoperability among devices. Middleware in this case abstracts over all the devices and communication technologies, and offers the applications well-defined interfaces to interact with other nodes.

Our aim is to enable interoperability among different nodes in MME without the need of adjustments each time new hardware is introduced. The agent-based approach offers an abstraction for the different hardware: it sees all the different nodes in the network as independent entities, which we call device-agents. Each device-agent knows its capabilities and constraints. Depending on its capabilities, a device-agent can offer services to other agents and can also perform tasks, sometimes using the services of other agents.

The complexity of agents representing, for example, a small sensor node or an unmanned vehicle (UV) can vary considerably. Thus, a sensor-agent might be only capable of measuring the current temperature and sending it to a receiver interested in this data, whereas the UV-agent can not only move through the environment, but also collect the data from the sensor-agents, aggregate and evaluate the collected data, and use this information further in its decision making process. Communication between different nodes is enabled by using a uniform message structure and by defining basic interaction mechanisms.

In MME we must handle different kinds of nodes not known in advance. The same applies for the network topology and for communication channels. Therefore an architecture is required which accommodates all these differences and is applicable for all possible participants in a heterogeneous network.

It this work we defined the Device-Agent based Middleware for Mixed Mode Environments (A3ME[1]) framework that enables ad-hoc interactions among heterogeneous devices on the fly. The framework is applicable to different usage areas. The heterogeneous devices are represented as device-agents and are enabled to describe themselves to others using a predefined extendable classification. The defined classification allows to classify the device type and types of capabilities. This enables the devices to work with classes of capabilities instead of specific capabilities, simplifying the interaction among different devices. To query and describe different types of information among devices a new query language has been defined. The structure of the messages is defined using Abstract Syntax Notation One (ASN.1) allowing to define their content dynamically. The use of ASN.1 Packed Encoding Rules allows to encode the messages in a very byte-length efficient way. As a proof of concept prototypes for a workstation, smartphone and different types of wireless sensors have been implemented.

1.1 Problem Statement

Today we have MMEs as described before almost everywhere:

- multimedia devices at home, at work and at public places,
- personal and mobile computers,
- smartphones and tablets,

[1] The name A3ME is an acronym for "Device-Agent based Middleware for Mixed Mode Environments" where "MMM" is abbreviated to "3M".

- wearable sensors and computers in our clothes,
- embedded sensors, computers and actuators in our cars,
- sensors in the environment,
- etc.

The problem is that many of these devices are island solutions, meaning they are not aware of other devices around them and are not able to interact with them. In example 1 we describe a typical situation for a mobile device of a person. It seems to be obvious that we need a way to enable these different devices to discover each other, and to enable interaction where reasonable. Some of these island solutions are being extended now to make the island bigger or to connect to other islands.

Example 1. The mobile device of a person is faced with different domains of heterogeneous devices. These domains can be home, car, office, shopping mall, etc. In some of these domains the interactions of the devices are already organized by some framework. At home there might be a smart home solution in place controlling the heating and the intrusion security. The car has its own framework for communication and infotainment. In the office some smart office framework may be in place, allowing to use the office infrastructure. For the specific domains there is a growing number of solutions already available. For interactions with the different domains the mobile device requires an own software for each of the solutions and each has to be configured individually. Often the situation is even worse: in each domain there exist multiple solution in parallel for different tasks, and each one requires its own software and configuration on the mobile device to interact with it. In the smart home the solutions for the heating and for the intrusion security might exist in parallel without the possibility of interactions between those.

To deal with the problem of enabling interactions among different devices we have to break it down into parts in the top-down manner:

1. The *interactions* among devices can be exchange of information or use of services.
2. To enable these interactions we first of all need to *enable the communication* between the devices.
3. To communicate the other *devices need to be discovered*.
4. Once the communication is established it is needed to *discover capabilities and services of other devices*.
5. The discovery of other devices capabilities and services requires each *device to be aware of its own capabilities*.
6. The exchange of device descriptions requires a *formal definition* for the *devices and capabilities descriptions*.

The identified problems are pillared by four issues:

- Communication,
- Device description,
- Interoperability and
- Security.

The *communication* has to deal with all kind of problems to deliver information from one device to another such as: discover the other device, address the devices, encode and decode the information transported, incorporate security, privacy and quality of service mechanisms. The *device description* enables the description of the devices, and their capabilities. The *interoperability issue* is interconnected with the communication and the device description. The *security issue* deals with the security and privacy of the information during transport, distribution and storage.

In this work we deal with the communication, device description and interoperability issues, but the security issue we only keep in mind and build in mechanisms to be used later by the concrete security solutions. The concrete security solutions are a huge research problem for its own and are not part of this work.

Our challenge is to enable and to simplify discovery and interaction between different devices in MME. When designing a middleware for MME in addition to challenges inherited in technologies present in MME we have to deal with new challenges introduced by the combination of the different dimensions of heterogeneity.

1.1.1 Challenges from WSNs

Since WSNs are part of MME we have to deal with all the problems already known from homogeneous WSNs. In [139] some of the requirements are summarized:

- appropriate abstractions and mechanisms for dealing with the heterogeneity of sensor nodes,
- provide a holistic view on both WSN and traditional networks,
- provide support for automatic configuration and error handling,
- support for time and location management.

Henricksen at al. [88] describes also further requirements to align WSNs with context aware systems:

- middleware solutions for WSNs must be more generic and assume heterogeneous sensor hardware and diverse communication mechanisms,
- use standard ontologies,

- align with standards.

Another major challenge for WSN is that the sensor nodes are usually very resource constrained in matters of: computing power, storage, energy and communication.

1.1.2 Challenges from Ubiquitous Environments

Ubiquitous environments are environments where Ubiquitous Computing as predicted in [156] has became reality. Ubiquitous Computing describes the situation where the computers (or communicating electronic devices) are everywhere in our surroundings. Another name for Ubiquitous Computing is also Ambient Intelligence [70]. The modern term Internet of Things (IoT) is becoming the realization of Ubiquitous Computing. IoT means that every thing in our surrounding is connected to the internet and is capable to interact with other smart things.

There are multiple areas which are Ubiquitous Environments:

- Smart Home,
- Smart Factory,
- Smart City,
- Connected car,
- etc.

All of this areas deal with heterogeneous devices which are connected and interact with each other.

The requirements identified together with the scenarios for Ambient Intelligence by the Institute for Prospective Technological Studies (IPTS)[2] in collaboration with DG (Directorate General) Information Society and with the active involvement of 35 experts from across Europe are [70]:

- Networks should be configurable on an ad-hoc basis according to a specific task with variable actors and components.
- Databases whether centralized or distributed should be accessible on demand from anywhere in the system.
- Wireless 'Plug and Play' solutions.
- Multi-domain networking.

The aim of this IPTS project was to describe what living with "Ambient Intelligence' might be like for ordinary people in 2010 [70]. Even though the technology has advanced a lot since these scenarios were developed in the year 2001, many of the individual requirements mentioned here are still not solved.

A middleware for Ubiquitous Environments has challenges which are overlapping with those for WSNs, but there are also some additional ones. Niemelä at al. [127] grouped the requirements for software in Ubiquitous Environments: interoperability, heterogeneity, mobility, survivability and security, adaptability, ability of self-organization, augmented reality and scalable content.

1.1.3 Challenges from Unmanned Vehicles Area

Joint Architecture for Unmanned Systems (JAUS) [140] (see also section 3.3.2) is an initiative to develop an architecture for the domain of Unmanned Systems. JAUS defined requirements for the architecture of Unmanned Vehicles (UV) to assure interoperability and adaptability to future technologies:

- Node platform independence: to accommodate platform evolution.
- Application independence: to satisfy multiple application domains.
- Technology independence: to handle multiple technologies and technology evolution.
- Decentralized approach: without a single point of failure.

These requirements also hold for MME since here we also have unmanned vehicles. We have to generalize those to fit for various possible nodes in a MME.

1.1.4 Heterogeneous Environment

In MME the combination of the different dimensions of heterogeneity (Figure 1) introduces additional challenges to a middleware.

1.1.4.1 Heterogeneity of Devices

The heterogeneity of devices available in MME introduces additional complexity to the middleware. We have to define mechanisms to discover other devices in MME and to exchange information with those. Therefore, the middleware needs an abstraction for all the different devices and has to enable interactions between those.

Enabling interactions requires definition of a query language and definition of data structures to exchange. For asking and answering the queries it is required to refer to the different capabilities and properties of the individual devices. For

[2] The Institute for Prospective Technological Studies (IPTS) is one of the seven scientific institutes of the European Commission's Joint Research Centre (JRC).

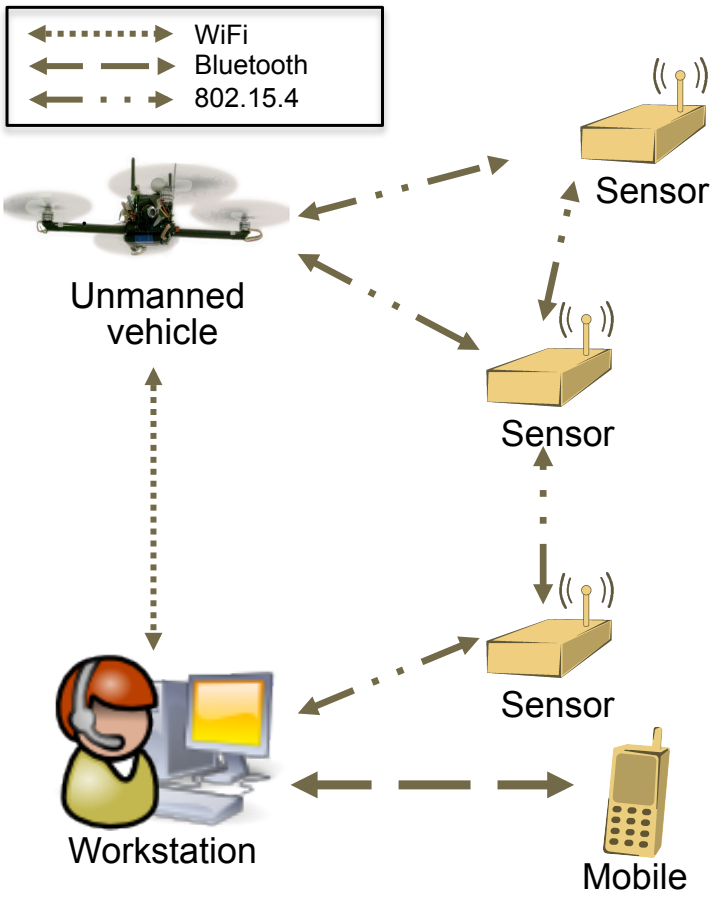


Figure 1: Dimensions of Heterogeneity

this the devices must be able to describe themselves and their capabilities to other nodes in the network. Theoretically, there can be an unlimited number of different devices with different capabilities and properties in MME.

Since a network in MME can not rely on the constant availability of individual nodes, all these mechanisms must work in a distributed way without a central coordinating instance.

1.1.4.2 Heterogeneity of Communication

The second kind of heterogeneity in MME is reflected by the variety of communication technologies used. This means there are various protocols and standards for different transmission media. Consequently, the properties of communication are quite diverse in terms of bandwidth, reliability, communication range, etc.

To deal with this communication heterogeneity, abstract mechanisms must be introduced which are independent of the individual communication technologies. The message structures and encoding schemata should be specified independently of the underlying communication media and technologies. This means we need a common basic encoding to be used for all the different devices.

The physical environment in MME can be harsh, so some nodes in the network may fail or lose their connectivity. Additionally, nodes can join and leave the network and some nodes may be mobile within the network. Therefore, the devices as nodes in a network can appear and disappear at any time. Furthermore, the presence of individual nodes, their positions, and capabilities are not always known in advance. This means that mechanisms for ad-hoc network setup, self-healing and self-configuration must be defined.

1.1.4.3 Heterogeneity of Software

The devices in MME can have very different types of operating systems, which is directly related to the type of device. But even for the same type of device usually there are multiple operating systems applicable to it. In the case of wireless sensor nodes the operating system is often compiled together with the program into a single binary and is then deployed to the node. This means there is no operating system as in the case of personal computers on top of which the programs can be installed and run.

Considering the program software the heterogeneity ranges from a single firmware like program per device to a set of thousands of programs running concurrently on a single device using a modern operating system or even running distributed on various devices.

The programs can be developed using the whole range of programming languages using various application programming interfaces and content representations.

1.1.5 Security and Privacy Issues

The growing number of interconnected devices increases the risk of the devices being compromised to disrupt or misuse their operation. The data exchanged among the devices might be a highly valued target for being stolen, changed or delayed. This makes it very important to design and build mechanisms to ensure security and protect privacy while enabling interactions among devices. The security and privacy topics are huge research areas on its own and not in the scope of this work.

1.2 Goals

The goal of this dissertation is to develop a framework which enables ad-hoc interactions among heterogeneous devices in a Mixed Mode Environment (MME) on the fly [91]. Hereby we do not mean just devices of similar type, but also very heterogeneous device types potentially from different areas. Our solution shall be applicable to different areas, technology islands and applications from beginning on. This means devices must be able to

- discover each other, without individual manual adjustment,
- exchange their descriptions and
- offer/use services to/of each other.

Hereby additionally the following principles are followed: decentralized solution, technology independence and generic solution.

1.2.1 Decentralized Solution

In a large MME with thousands of devices it is hardly possible to control the devices in centralized manner. Even in the cases where it is possible, the central control instance introduces a single point of failure for the whole system.

Therefore our goal is to enable ad-hoc interactions among the different devices in a decentralized manner without a central coordinator. This means there should not be any central instance needed to enable the interaction. This does not mean that there can not be one or multiple central instances for performance issues.

1.2.2 Technology Independence

Information technology evolves very fast and many specific technologies that are standards today will be obsolete in a few years. Therefore it is important to design a middleware that abstracts from the specifics of individual technologies and makes it possible to replace them or use different technologies in parallel.

1.2.3 Generic Solution

Nowadays many application areas are merging together or are at least interconnected. Therefore it is inconvenient to use different solutions for individual areas. To overcome this, our work shall offer a generic solution which is not restricted to a specific area.

1.3 Contributions

In this work we identified requirements for a framework, which must be fulfilled to enable ad-hoc interactions among devices varying in type and capabilities. According to the identified requirements we developed solution concepts, designed a framework and implemented prototypes for different devices. Figure 2 shows the contributions of this work and presents those in groups beginning at the top with the concept and continuing in steps to the bottom to the realization on different devices. The blue boxes show different conceptual elements developed in this work. The gray boxes show existing work reused and integrated in A3ME. The green boxes show the different implementations realized in this work to run on various devices represented by red boxes.

In detail, the contributions of this work include:

- **Device Interactions** (4.9): The A3ME framework enables basic interaction between heterogeneous devices in heterogeneous environments.
- **Device Description** (4.4): Description of the devices and their capabilities, properties and services by referencing them to a generic predefined A3ME classification.

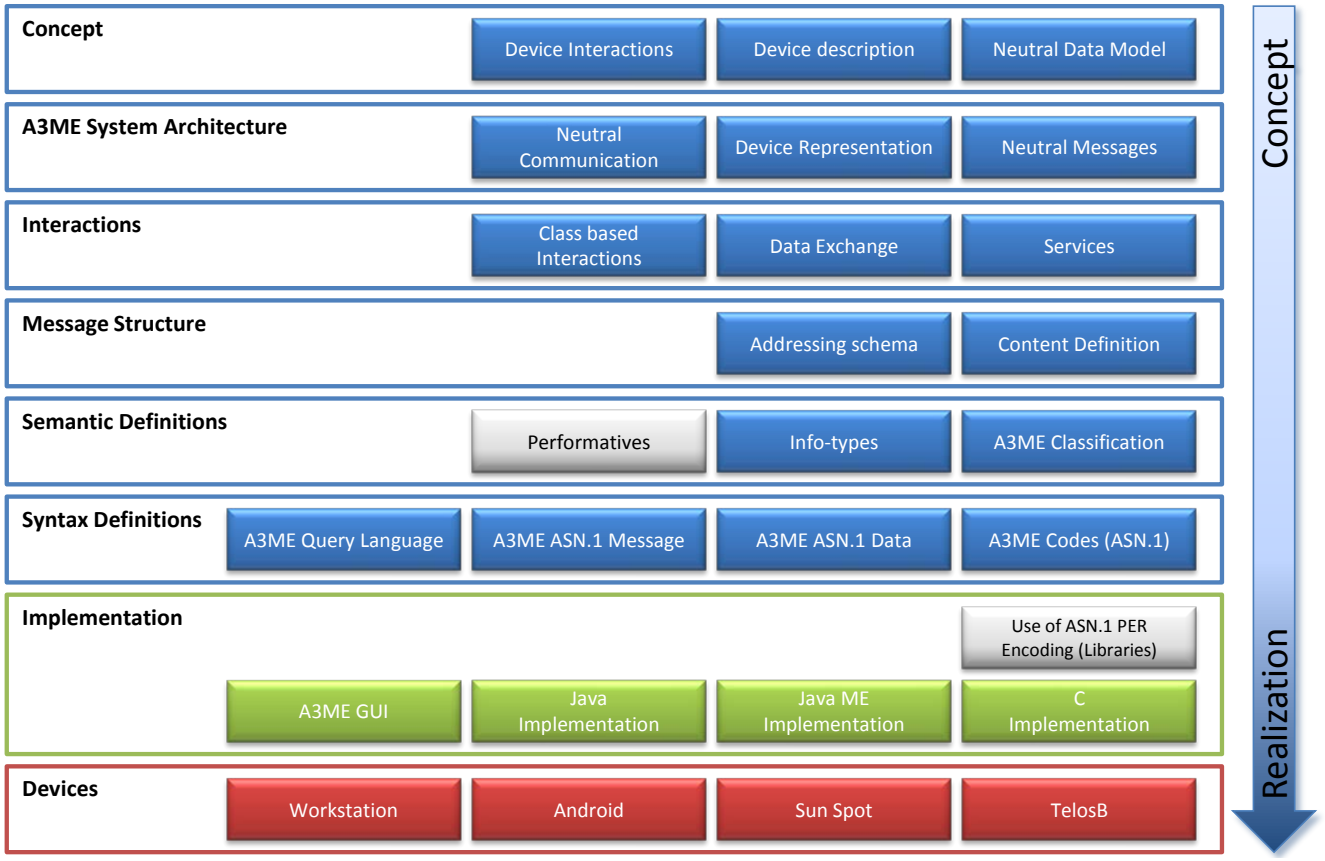


Figure 2: Overview of the A3ME Contributions (in blue and green color)

- **Neutral Data Model** (4.1.1): A3ME classification and data definitions offer a neutral data model for the different technologies. This also allows to translate the information from different technologies into the A3ME representation and back, which reduces the amount of required translations from $n*(n-1)$ when every technology needs to be matched to every other technology to n translations to the neutral A3ME representation.
- **Interconnections of Technologies**: **Neutral Data Model** allows to interconnect different existing communication technologies (4.5.5) and middleware solutions (4.5.6), and therefore extend them with more interoperability while being simple enough to be usable on resource constrained nodes directly.
- **Neutral Communication:** Definition of an interaction model which is independent of the used communication technology: by use of messages which can be transported on top of any communication technique (4.5).
- **Device Representation** (4.2): A novel way to deal with heterogeneous devices by abstracting and representing each device as a device-agent with capabilities and properties which can be exchanged with other device-agents.
- **Neutral Messages** (4.5.3): Use of technology independent messages to realize interactions between devices .
- **Interaction Types** (4.9): Definition of interactions for **Data Exchange** and **Service Invocation** by referencing the individual actions to **Performatives**.
- **Performatives** (4.8.1): Reuse of performatives concept introduced by FIPA (3.5.6) as FIPA Communicative Acts, which are assigned to each message according to its purpose.
- **Discovery** (4.10.4, 6): Basic interactions offered by A3ME device-agents enable ad-hoc device, capabilities and service discovery.
- **Class based Interactions** (4.9): Interactions are based on classes of capabilities and allow interaction with a priori unknown nodes.
- **Content Definition** (4.8.2, 4.10): A flexible and generic content definition usable for different areas and tasks.
- **Addressing Schema** (4.5.2): Definition of a technology independent addressing of devices by use of Device-Agent-IDs (DAIDs). The DAIDs contain different communication technology specific addresses of a concrete device.
- **Info-types** (4.10.2.1): Definition of information types to address different types of information.
- **A3ME classification** (4.7): Definition of the generic predefined extendable A3ME classification for description of the device types, capability types, capabilities, properties, services and data types.
- **A3ME Query Language** (4.11): Definition of a content description language **A3ME-QL** for a user-friendly definition of the messages for device interactions.

- **ASN.1 based Definitions** (4.10): Identified ASN.1 (Abstract Syntax Notation One, 3.8.5) standard in combination with ASN.1 Packed Encoding Rules (PER, 3.8.5.2) as best fitting solution for message and data definition and their encoding/decoding for transmission in the A3ME framework. The definitions are separated in:
  - ASN.1 Definition of the A3ME Classification (A.2),
  - ASN.1 Definition of the A3ME Message Parameters (A.3),
  - ASN.1 Definition of the A3ME Content Data (A.4),
  - ASN.1 Definition of the A3ME Object Identifiers (A.5).
- **Encoding/Decoding** (4.13): The use of ASN.1 PER (3.8.5.2) allows to encode/decode messages in a very byte-length efficient way. For the prototype implementation we used existing ASN.1 PER libraries.
- **Generic Data Structure:** Definition of flexible data structure for messages and data in ASN.1 allowing its use for different purpose and in different areas.
- ASN.1 based messages and data definition **allow to generate program code** for these structures, thereby simplifying and accelerating the development of device-agent implementations and applications for new devices.
- **Proof of concept: implementation of prototypes** for different types of devices (5). The implementation was not explicitly adjusted to enable the interactions among specific technologies. Instead the different parts were only implemented to interact with and through the neutral representation developed in this work. Therefore the achieved interactions were enabled through the connection of the different technologies via the neutral representation developed in this work.

2 Requirements for the Middleware in MME

From the goals (section 1.2) and from challenges (section 1.1) we identified the following groups of requirements for the middleware:

(R1) Self-description of devices (SD),

(R2) Technology independent interaction (TII),

(R3) Decentralized solution (DCS),

(R4) Applicable to Heterogeneous Environments (Het),

(R5) Low Hardware Requirements (LHW).

This requirement groups are described in the following sections.

2.1 R1 Self-description of Devices (SD)

Devices must be able to describe themselves to other devices when they discover each other. First the type of device and later the capabilities of the device must me described. This requires the use of a classification for the device types and for the capabilities, to match the type against. Here it is not feasible to use just a textual type description, since it can not be interpreted directly by machines. Even for humans the textual description in not always understandable, e.g. when a different language is used.

The described self-description can be done using a basic classification. This basic classification should meet the following requirements:

(R1a) **Device classification**:
For self-description of the devices a basic classification for different types of devices is required. The device types can be personal computers, smart phones, multimedia devices, sensor nodes, tags, etc.

(R1b) **Capabilities classification**:
For the description of the capabilities of a device a classification for the different capabilities is required. The capabilities should also be grouped into groups of similar capabilities.

(R1c) **Classification extension**:
Since a predefined classification can not be all-embracing it needs to be extendable. This means that it should be possible to further extend each classification element recursively into smaller more precise sub elements.

(R1d) **Implementation independent classification**:
The middleware should be able to describe these capabilities, independent of the software used to run or realize those capabilities.

2.2 R2 Technology Independent Interaction (TII)

One of the requirements for our work is to develop a framework which is technology independent. This means that the framework must be able to use various existing and future technologies. Technology independent interaction requirement can be subdivided into following sub requirements:

(R2a) **Technology independent communication primitives**
Communication primitives allow basic exchange of data (e.g. sending and receiving messages). The concrete realization used must offer the communication primitives, and it should be possible to replace the communication technique by a different one. It can be the case if the communication partners have more than one communication technique in common or if the communication component of a device is replaced by a different one.

(R2b) **Flexible technology independent message structure**:
To enable exchange of various types of information like queries, data, commands, etc. the message should be defined the way it can be transported on top of different communication technologies. It must be possible to define the message content dynamically allowing to contain different type and amount on information.

(R2c) **Common encoding and decoding schema for messages**:
To be able to transport a message on top of different communication techniques we need a common encoding and decoding schema to transform the message to and from byte-stream.

(R2d) **Technology independent interaction protocols**:
It is required to define a set of Interaction protocols for interactions among devices. This interactions can be exchange of request, answer and other messages.

(R2e) **Technology independent query language**:
To formulate and interpret the content of the various message types for device interactions a query language is required.

2.3 R3 Decentralized Solution (DCS)

To enable ad-hoc interactions among devices any two devices which support the middleware and can technically communicate with each other, must be able to discover and to interact with each other without the need for a third device. Two devices can technically communicate if they have a compatible communication component in common.

2.4 R4 Applicable to Heterogeneous Environments (Het)

Many existing middleware solutions are only applicable to partitions of devices available in a heterogeneous environment. Therefore we define three requirements for very different groups of devices to be able to check and to compare the different middleware solutions. The middleware must be applicable to the various areas of MME:

(R4a) **Applicable to WSN** (WSN)
Wireless sensor networks which are a continuously growing part of the heterogeneous environments have very specific requirements like operating with limited energy, communication, computing and storage resources.

(R4b) **Applicable to Ubiquitous Environments** (UbE)
Another important part of a MME are the Ubiquitous Environments like smart home or smart office. The middleware must be able to cover the requirements of such a Ubiquitous Environment.

(R4c) **Applicable to Unmanned Vehicles** (UxV)
Unmanned vehicles or robots are another part of MME. The middleware must be able to enable the interaction between this type of devices with other devices in the MME.

2.5 R5 Low Hardware Requirements (LHW)

Some of the devices in a MME are very resource constrained, therefore it is necessary that the middleware has very low minimal hardware requirements to be usable by a device.

Many sensor devices have an 8 bit microcontroller, a very limited amount of program and storage memory, and are powered by a battery. This means the computing and storage capabilities are very limited.

Additionally the communication bandwidth and reliability on these devices is very limited too. We also have to consider that communication is the most expensive part on sensor nodes with respect to energy consumption. Often it is more efficient to spend more computing power to compress or encode the message more compactly to reduce the amount of data to be sent/received.

These facts dictate a very important challenge for our work, namely to keep the foot print of the framework low to allow the resource constrained devices also to use the framework directly.

In this work we consider the TelosB platform (see 5.6.1) as the minimum hardware requirement, but this does not mean that it is not possible to use the framework even for less powerful devices.

2.6 Use Cases

To concretize this requirement we define a set of use cases describing different type of tasks the middleware shall support. We assume that the following types of devices are present for the use cases:

- simple sensors (e.g. temperature, motion, light),
- complex sensors (e.g. video camera, biometric identification sensor),
- actuators (e.g. light controller, window blinds),
- user interfaces (e.g. switches, device controller),
- smartphones,
- unmanned vehicles,
- workstations,
- servers.

For all use cases the precondition is assumed that any pair of devices required to interact for a specific use case either use the same communication technology or a communication path exists (e.g. using bridges), which allows them to communicate. This might require one or multiple bridging device(s) in between, which forward the messages between different communication interfaces. If a device gets a request from a specific device over multiple communication interfaces, it should use only one communication interface to answer. The selected communication interface should be the most efficient one in terms of cost out from the set of available communication interfaces which still fulfill the required quality of service requirements to answer/complete the request. The cost criteria can be monetary, energy consumption, communication bandwidth, computing, storage, etc. or a weighted combination of those.

2.6.1 Use Case UC1: Device Discovery

The basic requirement is to be able to discover various kinds of devices which are around a user or a device. A device needs to find out which other devices are in communication range (not necessary direct).

Primary Actor:	A device or a user with a device
Precondition:	The current device and devices to be found use the same communication technology or a communication path exists (e.g. using bridges), which allows them to communicate.
Trigger:	- A device is switched on, - A device enters a new area, - An application/user request for devices in range.
Extensions:	- Find devices of a specific type, - Find devices with specific capabilities (e.g. with a temperature sensor), - Find devices with specific properties (spatial, functional, non-functional).

2.6.2 Use Case UC2: Information Query from Other Devices

A device needs to collect information from other devices. This information can be about capabilities, properties, data values, names or descriptions.

Primary Actor:	A device or a user with a device
Precondition:	The current device and devices to be found use the same communication technology or there exists a communication path (e.g. using bridges), which allows them to communicate.
Trigger:	- A new device is discovered, - A new task requires to update the knowledge about other devices, - A corresponding application/user request.
Extensions:	- Add a condition to be satisfied for the query.

2.6.3 Use Case UC3: Service Discovery

A device needs to find services offered by other devices.

Primary Actor:	A device or a user with a device
Precondition:	The current device and devices to be found use the same communication technology or there exists a communication path (e.g. using bridges), which allows them to communicate.
Trigger:	- A new task requires the knowledge about services offered, - A corresponding application/user request.
Extensions:	- Find services of a specific type, - Find services related to specific device capabilities, - Add a condition to be satisfied for capabilities/properties.

2.6.4 Use Case UC4: Call Simple Core Services

Use a core service offered by another device.

Primary Actor:	A device or a user with a device
Precondition:	A service offered by another device was discovered before.
Trigger:	Service call request through a user or an application.
Extensions:	- Invoke the service periodically every given time-value. - Invoke the service for a period of time.

2.6.5 Use Case UC5: Call Other Established Services

A device discovers a service through a local network, but the service is only offered through another communication interface on a different network with less limitations (e.g. in bandwidth). Since the actual device is also connected to the second network, it can start using this service.

Find information about services offered through another interfaces and then invoke the service through it. For example discover that another device found through a WSN is offering a Web service and then start using it directly (not through the WSN).

Primary Actor:	A device or a user with a device
Precondition:	Some devices (service provider) are offering established services (e.g. web services). The primary actor and service provider are connected through a local network (e.g. WSN) and additionally to a second network (e.g. internet) through which the service is offered.
Trigger:	Discovery of other service the current device wants to use.

3 Related Work

In this work we developed a framework to enable interactions of devices which usually belong to different research areas. Therefore we first discuss related work from specialized areas: WSN (3.1), WSANs (3.1.6), Ubiquitous Environments (3.2) and robotics (3.3). Afterwards we discuss specialized middleware related technologies (3.4) and generic middleware solutions (3.5).

After looking at solutions for different types of devices we have to consider technologies related to interoperability. This are first technologies for generic data definition and serialization/deserialization (3.8). To decide how to represent the content of the messages, which the interacting devices have to exchange, we discuss various content description languages in section 3.9.

For exchange of descriptions of devices and their capabilities these descriptions should be based on a common classification or ontology. In section 3.10 we present technologies related to this topic.

The framework developed here shall not only enable machine to machine (M2M) interaction, but also human interactions with these machines. The ASN.1 representation of messages and content we use in our framework has many advantages for the M2M interactions, but is difficult to be read and written by humans. Therefore as part of this work we developed a new content query language. In section 3.11 we discuss a few technologies related to content query languages.

3.1 Frameworks for Wireless Sensor (and Actor) Networks

The wireless sensor networks represent the most resource constrained part of the devices in MME. A variety of different middleware solutions for Wireless Sensor Networks (WSNs) already exists. Henricksen et al. [88] give an overview of the different approaches for middleware for WSNs and group them into categories:

- database-inspired approaches,
- tuple-space approaches,
- event-based approaches.

Not included in this categorization are

- virtual machine approaches.

Another overview of middleware in WSNs is given in [86]. These two survey papers provide an overview over the existing middleware approaches in WSN.

3.1.1 Operating Systems for WSN

Most middleware solutions for WSNs are closely related to the underlying operating system (OS). The operating systems for sensor nodes are quite different from the conventional operating systems, as they are mostly not set-up on the hardware a-priori, but compiled together with the application and are then deployed to the sensor nodes.

3.1.1.1 TinyOS

TinyOS[3] is a lightweight open-source operating system designed for resource-constrained wireless sensors. It simplifies the programming of WSN nodes by providing a set of important services and abstractions, such as sensing, communication, storage, and timers. [110]

TinyOS is built and used with the NesC. NesC extends the programming language C with a component-based architecture, where the components are connected via bidirectional interfaces. [79]

TinyOS is used and maintained by developers and users from industry and academia worldwide. It supports a variety of platforms, sensors and protocols.

In our work we used TinyOS to implement the early A3ME software prototypes for the TelosB sensors, but had to switch later to the Contiki OS.

3.1.1.2 Contiki

Contiki OS[4] is an open source, multi-tasking operating system for networked embedded devices and WSNs. It is optimized for resource-constrained devices. Many key mechanisms and ideas from Contiki have been widely adopted in the industry. Contiki is based on an event-driven kernel but provides support for both multi-threading and a lightweight stackless thread-like construct called protothreads. Contiki contains communication stacks for IPv4, IPv6, 6LoWPAN and Rime. Rime is a lightweight communication stack optimized for WSNs. [32]

Contiki software is programmed using standard C. The provided simulator allows to test the compiled software and its interactions in a network before deploying it to the hardware. Many libraries from Contiki and from standard C greatly simplify the development efforts.

We use the Contiki OS for the development of the A3ME prototype for the TelosB platform (section 5.6).

[3] TinyOS open source project collaboration web page: https://code.google.com/p/tinyos-main/.

[4] Contiki operating system home page: http://www.contiki-os.org/

3.1.2 Database-inspired Approaches

In database-inspired approaches the wireless sensor network is seen as a database. Two famous works in this area are Cougar [159] and TinyDB [115]. In Cougar and TinyDB the WSN is treated as a relation, where the sensor readings for each instant in time are represented as a row. The columns contain the attributes e.g. sensor readings, the sensor id, the reading number (epoch), etc. The information from sensors is queried by formulating queries in an SQL like query language. In a SELECT-FROM-WHERE clause the attributes and the conditions to be queried can be specified. The results can also be grouped in the GROUP-BY part and aggregated in the SELECT part of the query. For queries which have to be repeated periodically the repetition period and the duration can be specified. An example query is shown in listing 1.

```
SELECT nodeid, light, temp
FROM   sensors
SAMPLE PERIOD 1s FOR 10s
```

Listing 1: TinyDB Example Query

We also reimplemented the TinyDB concept in a student project and later on as an internal project. Our implementation was done for the TelosB sensor nodes using the Contiki operating system (see section 3.1.1.2). The results of this project are published in [98]. This project is not a direct part of this dissertation, but it inspired us to the information type queries (section 4.11).

3.1.3 Tuple-space Approaches

Tuple-space approaches use key-value tuples to store and access information, building this way a type of shared memory among the nodes. TinyLIME [62] uses this approach to store and access sensor values locally and on neighbor nodes.

We don't use this approach in our work to realize the core functionality of our framework. Nevertheless it might be a reasonable service offered across different devices on top of A3ME to share information.

3.1.4 Event-based Approaches

The primary task of the WSNs – monitoring physical phenomena and report the observations – makes the event-based approach very reasonable to be used here. Listing 2 shows a typical event definition in an event-based WSN. According to this definition a "fire' event occurs whenever the light exceeds the value of 100 lux and the temperature exceeds the 50 degree Celsius in a time window of 60 seconds.

```
EVENT fire =:: light > 100 AND temperature > 50 WINDOW 60 sec
```

Listing 2: Event-based Example Query

In an event-based solution it is required to define:

- *events:* simple events e.g. sensor readings and complex events like the fire event from listing 2,
- *a publish-subscribe mechanism* to register interest for specific events and to distribute the events if they occur.

For the definition of events issues like time stamping, time synchronization, position information, ordering of events, sliding window evaluation of the incoming event streams, etc. must be considered. For the publish-subscribe mechanism broker(s) need to be introduced and mechanisms for dissemination of the events need to be set up.

Many of the topics mentioned are hard to solve on unreliable resource constrained nodes in WSN, and are still hot research topics. ukuFlow[5] [85] is an ongoing project in this area.

The middleware which we describe in this dissertation provides the basic functionalities on top of which such higher level event-based services could be built. For example, for the definition of events it is required to know what type of simple events are available. Here the A3ME middleware, described in this thesis, could be used to query for the various sensor types.

3.1.5 Virtual Machine Approaches

In the virtual machine based approach some program code together with its execution state and variables can move from one node to another and continue its execution there. This combination of executable code, execution state and variables is often called mobile agent. To enable this approach the different hosts need to provide the virtual machine, which is able to host the mobile agents and to allow them to execute their code. Examples of this approach in WSN are Mate [109] and Agilla [76].

Agilla is a mobile agent middleware for sensor networks. A mobile agent here is a special program code together with state variables which can migrate and be duplicated across sensors. The agents use neighbor lists and tuple spaces for

[5] Research project from Pablo Guerrero at Databases and Distributed Systems Group, Technische Universität Darmstadt

coordination. An application here is usually composed from multiple mobile agents each one responsible for a specific task. [76]

3.1.6 Wireless Sensor and Actor Networks (WSANs)

Wireless sensor and actor networks (WSANs) [44] distinguish between two kinds of nodes: sensors and actuators. Sensors are seen as nodes with limited energy resource and reduced communication capability, whereas actuators have more energy, better communication capability and can interact with the environment, for example they can switch on or off some device.

3.1.7 SYLPH

SYLPH (Service laYers over Light PHysical devices) is a distributed service-oriented architecture for integration of heterogeneous WSNs into Ambient Intelligence environments. This work adds a service layer on top of different WSN solutions and is integrated with a multi-agent framework enabling the agents to use the information collected by the WSNs. [149]

Similar to our work SYLPH allows to offer services and service directories directly on the resource-constrained devices independent of the communication technology used. An own interface definition language (IDL) allows to describe and use services in a very byte-length efficient way. The disadvantage here is that a proprietary solution is used with static structures for the messages. In A3ME we use ASN.1 based definitions for the messages, allowing dynamic message structures, which also result in a very byte-length efficient payloads with ASN.1 packed encoding rules applied.

The main difference to our work is, that SYLPH focuses specifically on WSNs while our solution is applicable to various types of electronic devices and not just for wireless sensor nodes.

3.2 Frameworks for Ubiquitous Environments

Ubiquitous Environments are environments with many heterogeneous devices: sensors, actuators, human interface devices in humans surroundings. Projects in this research area are often focused to specific scenarios like: smart home, smart office, smart factory etc.

3.2.1 MIT's Oxygen Project

In the project Oxygen[6] different devices like handheld devices, actuator and sensors in an office are combined with different technologies to test and demonstrate the possibilities, achievable with already available technology. The users were enabled to interact with the system through voice or visual recognition. Context information and automation should be used to support the users in their tasks.

The devices had to automatically discover each other and provide the user with information and services at best effort an any time. The Oxygen Project formed the perspective to see the smart devices in people's surroundings and services these devices provide as a new type of resource: like the electrical power or network connectivity, which can be used for different tasks.

Many of the project goals are similar to ours, but the project is discontinued since 2004 after a set of demonstrations has been published. The ideas developed in the Oxygen project show the usefulness of a framework like A3ME, which will simplify the development of these ideas into demonstrators and products.

3.2.2 Mundo

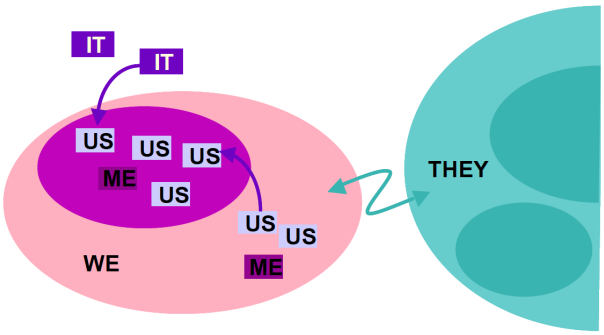


Figure 3: The Elements of the MUNDO Architecture [124]

[6] Web site of MIT's Oxygen project: http://oxygen.lcs.mit.edu/

Mundo architecture [124] [43] is an approach in Ubiquitous Computing, which deals with heterogeneous devices. It classifies the devices into five groups (Figure 3), according to their roles:

- ME (**M**inimal **E**ntity),
- US (**U**biquitous a**S**sociable object),
- IT (smart **IT**em),
- WE (**W**ireless group **E**nvironment),
- THEY (**T**elecooperative **H**ierarchical ov**E**rla**Y**).

All communication here is based on publish/subscribe scheme and is bundled into Mundo-Core communication middleware [42]. The physical communication is handled by transport services, which offer following basic operations: subscribe, unsubscribe, received, advertise, unadvertise and send. MundoCore has been implemented in Java, C++ and Python for PCs, PDAs and smartphones. [43]

While Mundo has its focus on smart home and office environments, A3ME has a more generic approach. It would be possible to combine the two approaches: the A3ME framework could be used to discover the various devices in an smart environment and collect their capabilities and Mundo could use this information to classify and assign the discovered devices into the five groups.

3.2.3 OpenHAB

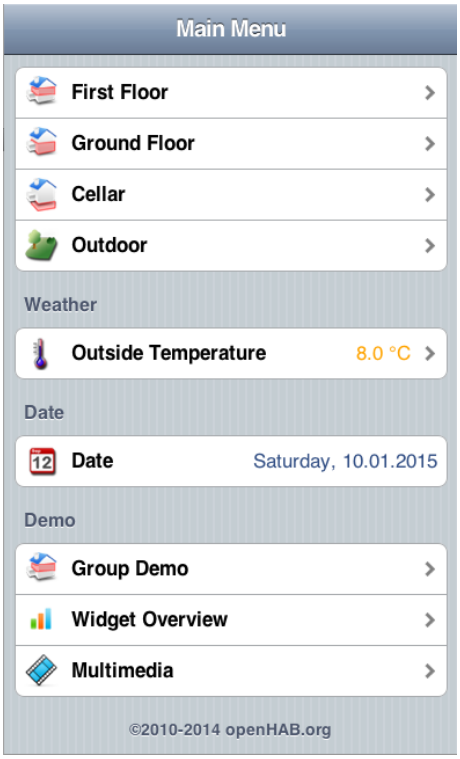


Figure 4: OpenHAB Web User Interface (Source: screenshot from demo server)

The open Home Automation Bus (openHAB)[7] project aims at providing a universal integration platform for all things around home automation. It is designed to be vendor, hardware and protocol independent. OpenHAB brings together different bus systems, hardware devices and interface protocols by dedicated bindings. These bindings send and receive commands and status updates on the openHAB event bus. This concept allows designing personalized user interfaces with the possibility to operate devices based on different technologies (Figure 4). [103]

While openHAB focuses on the integrations of different solutions to offer users a single interface, our work focuses on the definition of a new way to deal with the heterogeneous devices and the way they interact with each other.

3.2.4 Gaia

Gaia is a middleware meta-operating system for Infrastructure-based Large-Scale Pervasive Systems. The Gaia operating system treats a collection of devices, such as a smart room, as analogous to a computer by providing a well-defined abstraction to the various devices and services in the ensemble (Figure 5). Applications can be designed to that abstraction

[7] OpenHAB project website http://code.google.com/p/openhab/.

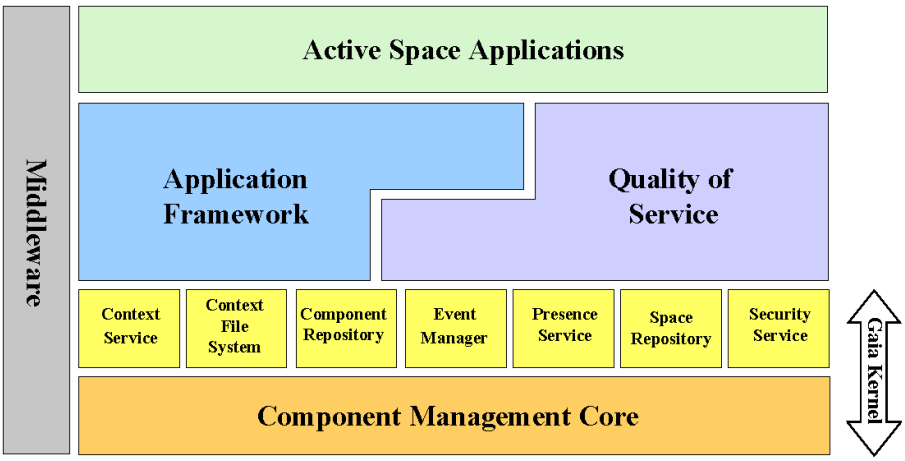


Figure 5: Gaia Architecture (Source: http://gaia.cs.uiuc.edu/)

without any knowledge of the underlying devices or services. A physically-bounded space of smart devices and applications managed by Gaia is called an Active Space. The purpose of such an Active Space was to support enable user-centric application. [138] [45]

A3ME could be used to discover the devices and their capabilities to build such an Active Space. The project is discontinued since 2005 and the project web page is not available anymore.

3.2.5 MIMOSA

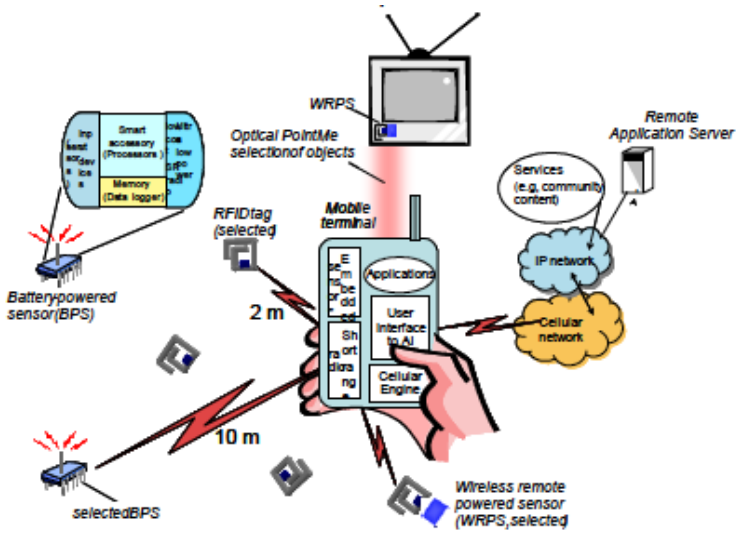


Figure 6: MIMOSA Framework [107]

MIcrosystems platform for MObile Services and Applications (MIMOSA)[8] was a project from the Sixth Framework Program for Research and Technological Development (FP6) of the European Commission from 2002 till 2006. The goal of the MIMOSA project was to develop an open technology platform for Ambient Intelligence with user's mobile device as the central user interface (Figure 6). [107]

The MIMOSA platform differentiates between five types of physical entities:

- Mobile Terminal,
- Remote Application Server,
- Sensor Radio Node,
- Active RFID node and
- Passive RFID TAG.

[8] Web page of the MIMOSA project: http://www.mimosa-fp6.com/.

Similar to A3ME the devices are seen here as entities. But in our framework the device types are more generic and allow to apply the framework to a broader range of devices, scenarios and applications. The MIMOSA framework is build around the user's mobile devices as user interfaces. Our framework also uses the mobile devices as user interfaces but they don't have a central role in A3ME. In contrast to MIMOSA the A3ME framework is device centric and allows also pure machine to machine (M2M) interactions with no user in the loop.

3.2.6 An Ambient Intelligent Platform based on Multi-Agent System

The Ambient Intelligent Platform based on Multi-Agent System is an approach which combines GAIA and AUML methodologies. The agents are modeled and assigned with roles according to the expected functionality. The agents themselves are defined via diagrams in Agent UML (AUML) methodology. [154]

This work focuses on the modeling and definition of the software agents for a multi-agent based solution for the ambient environment. It enables interactions between abstract entities like functionalities and tasks. The A3ME framework operates on a lower abstraction layer. It enables interactions between physical entities represented by electronic devices. Therefore it would be possible to apply the framework described in this section on top of A3ME.

3.2.7 uID-CoAP Architecture

The uID-CoAP architecture is a framework for embedded solutions. It combines Ubiquitous ID (uID) architecture with the constrained application protocol (CoAP) and offers the functionality of the nodes as RESTful services. The *ucodes* are used as unique identifiers for devices and specific services. The uID database in the backend provides semantical information and stores the data for the resources identified by ucodes. CoAP is a lightweight HTTP like protocol based on User Datagram Protocol (UDP) usually used together with 6LoWPAN over 802.15.4 network. [160]

In contrast to our work the devices here depend on the semantic information from the uID-database in the backend and can not start interactions if that information is not available. The described implementation of the uID-CoAP is build on top of a specific real-time operating system ITRON making it a very specific solution. The use of uIDs and CoAP will be considered to be used in A3ME and are discussed in section 4.5.2.

3.2.8 AllJoyn

AllJoyn is an open source project supported by the industry to develop a software and service framework to interconnect different devices. The project is hosted by the AllSeen Alliance[9] – a collaborative project of the Linux Foundation – with over 140 member companies. The AllJoyn project offers a shared code base and a common communication protocol. [46]

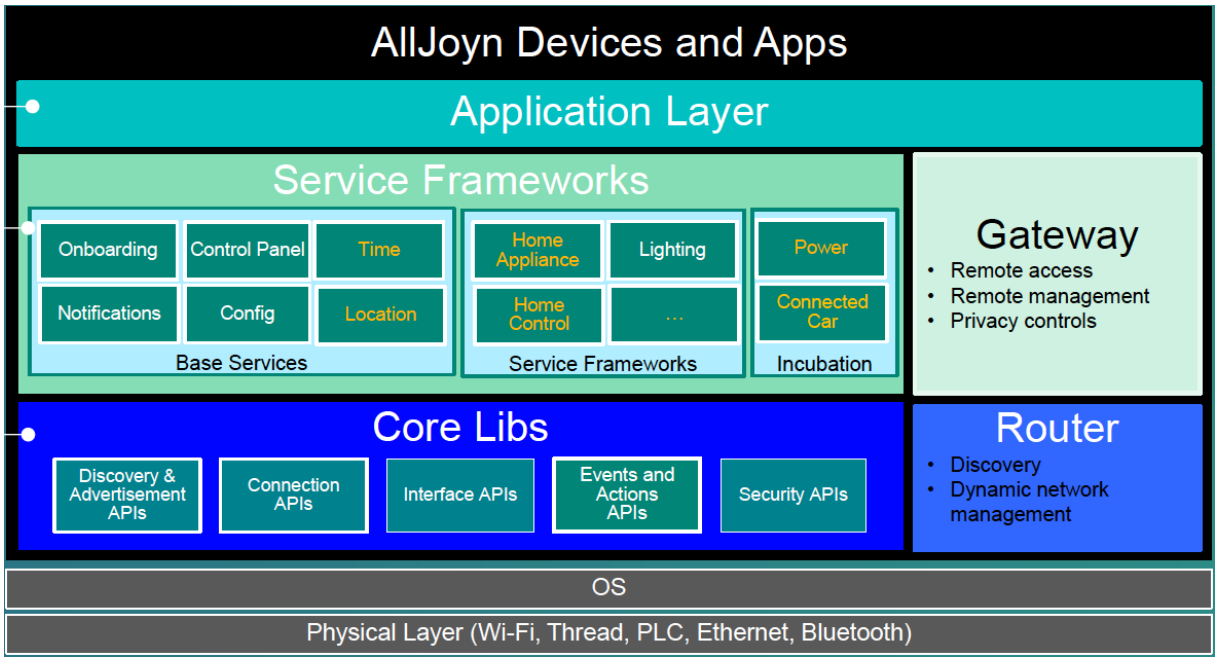


Figure 7: AllJoyn High-level Architecture [46]

[9] Web page of the AllSeen Alliance: https://allseenalliance.org/.

The framework distinguishes two kind of devices: rich devices with a high level operating system and resource constrained devices with a real-time operating system. All devices can have applications running on them which can offer and use services. The applications communicate with each other via published Application Programing Interfaces (APIs) using the ”distributed software bus“. The bus is established ad-hoc by discovering and linking to other services. It uses a wire protocol based on the D-Bus wire-protocol and can transport the encoded messages on top of different transport protocols. The nodes in the bus can be ”Routing nodes“ (RN) and ”Leaf nodes“ (LN). LNs can only connect to RNs, RNs can also connect to other RNs. The resource constrained devices can only be implemented as LN and can therefore only be connected to the AllJoyn bus via an RN.

The AllJoyn framework offers core libraries for different programming languages and services frameworks for basic functionalities and specific areas like for home control, lightning etc (Figure 7).

Similar to our framework the AllJoyn framework is build to enable ad-hoc interactions between heterogeneous devices. In contrast to our framework the ad-hoc functionality in AllJoyn framework is only possible for devices which implement the Routing Node functionality. Devices which only have the Leaf Node functionality and resource constrained devices, which can only be Leaf Nodes, can only interact directly with Routing Nodes or require a Routing Node to interact with other devices. See also the comparison in section 6.4.

3.3 Frameworks for Unmanned Vehicles and Robotics

Another class of devices in MME is represented by robotics and unmanned vehicles (UV). This type of devices are usually complex systems with numbers of sensors and actuators. They can be remotely controlled or operate autonomously. Usually they have reasonable computing, storage and communication capabilities.

3.3.1 ROS

Robot Operating System (ROS) is an open-source framework for robot software development providing operating system like functionality. It provides additionally to the services usually offered by an operating system abstraction for the different software, hardware and communication components of a robot. It offers mechanisms for interactions of these components and their management. The framework provides tools and libraries for obtaining, building, writing, and running code across multiple computers. [128]

ROS originated at Stanford Artificial Intelligence Lab and was further developed at Willow Garage. It has a very broad user base and is used in many leading robot research groups.

Core concept of ROS is its modularization. Each module is responsible for a well defined task and is loosely coupled with other modules. Modules can be grouped into bundles called stacks. The coupling between the modules can be done via topic-based publish-subscribe mechanisms, via message exchange or via services offered and used to/by each other. This modularization facilitates and simplifies the reuse of code and allows the framework to provide a lot of ready-to-use modules for robot developers and researchers.

We implemented an A3ME module for ROS (section 5.5), which offers other ROS modules an interface to interact with the A3ME framework.

3.3.2 JAUS

The Joint Architecture for Unmanned Systems (JAUS) is a set of standards for unmanned vehicle systems. It defines the component based system architecture for the individual UVs, their interactions witch each other and the communication protocols. JAUS uses Service Oriented Architecture (SOA) approach to control the UVs. The JAUS standards are owned and developed by the Society of Automotive Engineers (SAE) under the Aerospace Standards Unmanned Systems Steering Committee (AS-4). The purpose of JAUS is to make unmanned systems and robotics products from different manufactures more interoperable, exchangeable, and modular. [140]

JAUS systems are composed of a number of subsystems which represent a physical entity, such as an unmanned vehicle or operator control unit. Subsystems are composed of nodes, which are computing end-points, and each node can contain one or multiple components. Each component offers its functionality as services to other components.

The SAE published the JAUS standard as sets of related but separate documents [140]:

- JAUS Transport Specification (AS5669A),
- JAUS Service Interface Definition Language (AS5684),
- JAUS Core Service Set (AS5710A),
- JAUS Mobility Service Set (AS6009),
- JAUS Human Machine Interface Service Set (AS6040),
- JAUS Manipulation Service Set (AS6057).

The JAUS Service Interface Definition Language (JSIDL) defines the interfaces and the messages for the message oriented interactions for the JAUS components. XML schemata are used to validate the definitions. The JAUS Core Service Set (JSS Core) is a set of common services to enable:

- transport of messages between components,
- setup events,
- manage access control,
- manage the state of the components,
- exchange time representation,
- liveness check of a component (like ping),
- component discovery.

JAUS specifies the system architecture and the interactions specifically for unmanned vehicles. A3ME aims to enable interactions of different type of devices and does not specify the internal architecture of individual devices, but only specifies their interfaces to interact with other devices.

3.4 Other Specialized Frameworks

In this section we describe other specialized frameworks related to this dissertation.

3.4.1 QoS-aware Middleware for Ubiquitous and Heterogeneous Environments

The QoS-aware middleware for ubiquitous and heterogeneous environments [125] focuses on identification and provision of quality of service (QoS) required by the applications. The solution models the applications as collections of components with local and remote dependencies. For this dependencies QoS are specified and translated into profiles at deployment and are used at runtime to setup and adjust connections between components to meet the required QoS criteria.

In our work we focus among others on self-description capabilities of devices. With regard to this QoS-aware middleware uses (self-)descriptions related to required and provided QoS of devices and services.

3.4.2 ContextFramework.KOM

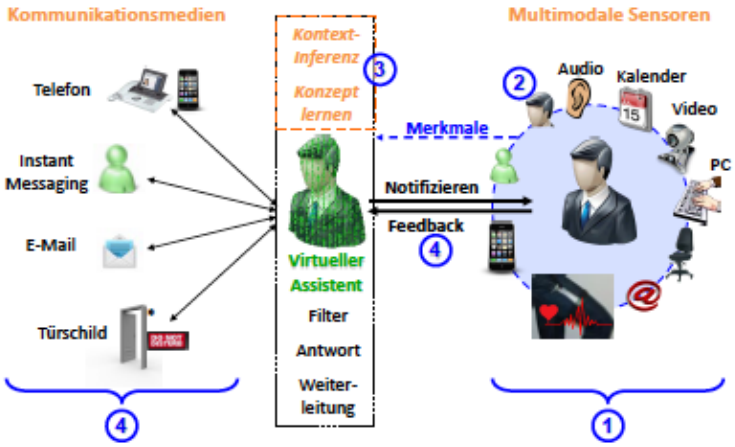


Figure 8: User Communication Assistant based on the ContextFramework.KOM [104] (1 - user and its context, 2 - information sources, 3 - virtual assistant, 4 - communication channels)

ContextFramework.KOM [143] [142] [104] is a middleware for integration of heterogeneous sensors and other information sources related to a person. Other information sources can be for example the location, a contact list, online calendar, etc. The system allows to evaluate the collected information to infer the context of a person and can be trained to trigger appropriate actions/services for a given context.

For new tasks to be supported by the system new information sources might be required. It is possible to search for this information sources and services. In addition to the type of information/service to be searched it is possible also to specify further parameters like Quality of Service (QoS) and Quality of Information (QoI), which have to be satisfied. The sensors and their capabilities are semantically described using Web Ontology Language (OWL) allowing to integrate new sensors.

The system is build on top of an OSGI framework. This allows to use the ContextFramework.KOM on many different platforms. On the other side it also introduces constrains: it requires a Java Virtual Machine and enough resources to run the OSGI framework and the required services. More constrained devices like sensors are connected through gateways. For realization of interactions following interfaces (connectors) are used:

- LOCAL: for services in the same Java VM,

- R-OSGI: for services between OSGI bundles,
- RMI: Remote Method Invocation for services in other Java VMs,
- XML-RPC: for services on non Java platforms.

The primary application supported by ContextFramework.KOM is an automatic user assistant (figure 8), which supports the user in managing the different communication channels depending on the current context of the user.

Compared to our work the ContextFramework.KOM middleware is focused on the integration of heterogeneous devices and information sources to determine the users current context. Our framework has a more generic goal: integration of all type of communicating devices and enabling their interactions among each other.

3.4.3 Speakeasy

The Speakeasy approach [72] developed at Palo Alto Research Center offers a set of protocols to enable users to interact with heterogeneous devices around them. Speakeasy approach focuses on three premises:

1. Use of small fixed set of generic interfaces to interact with other devices,
2. Use of mobile code to extend capabilities of devices if required and
3. All Interactions have to be triggered or allowed interactively by the user.

The first premise to use small fixed generic interfaces is similar to the solution we describe in this work, but while in speakeasy the interfaces are applied inside the communication itself, we abstract over the specific communication components by defining an generic communication interface. By doing so we are decoupling the interactions from the specific communication technology and allow at the same time a device to have and use multiple communication components.

The second premise of using mobile code to extend capabilities of a device to enable it to use a specific service requires the device to have a specific virtual machine, where the mobile code can be executed, and it also requires to transmit the mobile code itself which might exceed the communication and storage capabilities of a device. Therefore this premise contradicts to our goal of being usable on resource constrained devices.

The third premise requires all device interactions to be user-centric and excludes all device to device interactions where no user is involved. The goal of our work is to develop a framework where all devices can interact: the user-centric and those where no user is involved.

3.4.4 Continuum Architecture

The Continuum Architecture [68] [67] is a software architecture which introduces an indirection between the users, the software, and the software/service providers to provide the user with contextual services. These services are often not known a priori. The software is modularized here into self-contained units with clearly defined functionality (*frames*), which shall simplify the users interactions with contextual services and related data.

This solution provides the user with services available at her current location which are related to her current activity, e.g. when reading a document on her mobile device, she can display it on the screen nearby or print it on the printer available there. This functionality is enabled by reusing the functionality the users device already has – namely a web browser capable to use e.g. Java applets. The goal of this work was to avoid the installation of any special software explicitly. In the case of Java applets the required software is downloaded when the corresponding service is triggered by the user.

The Continuum Architecture research targets a similarly heterogeneous environment (they call it chaotic environments) as we do in our work but it focuses on users interactions with contextual services, while our work aims to enable and simplify the interactions of electronic devices in general. This related research especially points out the challenge of enabling interactions for distinct administrative and technological realms. Similar to this work we reuse device and service discovery protocols which exist in various technological and specialized area solutions.

3.4.5 ISO/IEEE 11073 Medical / Health Device Communication Standards

ISO/IEEE 11073 standards define the communication protocols, messages and data formats for medical devices. The goals of these standards are to enable ad-hoc interoperability for medical devices connected to patients and enable the exchange of data acquired by these devices. [11]

Within the ISO/IEEE 11073 family of standards Personal Health Data (PHD) standards [12] address the interoperability of personal health devices (PHDs).

All data definitions are defined using Abstract Syntax Notation One (ASN.1) (section 3.8.5). For efficient encoding/decoding of data to and from byte streams a specialized version of ASN.1 PER (see section 3.8.5.2) is used. The standards are defined independent of the communication technic used to communicate with other devices. So far there are three communication possibilities offered: Bluetooth, USB and ZigBee.

ISO/IEEE 11073 family of standard offers a solution with similar goals as our work with the difference that these standards are specifically for medical devices. Our goal in contrary is to offer a generic solution which is not restricted to a specific area.

3.4.6 Tsunami Service Bus

Tsunami Service Bus (TSB) [75] is an integration platform for heterogeneous sensor systems in GITEWS (German Indonesian Tsunami Early Warning System). It's main goal is to deliver a reliable tsunami warning message as quickly as possible. The system combines the information collected by heterogeneous sensors and sensor systems. These combined data allows to predict tsunami waves. TSB is realized as a SOA and implements the SensorWeb Enablement (SWE) standards and services (see section 3.10.2).

TSB is a specific solution for a system, which corresponds to the type of systems targeted by the A3ME framework. It has to deal with very heterogeneous devices and technologies which also continue to evolve and to be changed. In contrast to A3ME, where the interactions among devices have to be enabled, the TSB focuses on the integration and evaluation of the collected data.

3.5 Generic Middleware Solutions

Most existing generic middleware solutions are for conventional computers and have their focus on the distributed computing and on communication issues. The goal of our work in contrast is to enable interactions between different devices at all. Some of the technologies discussed in this section can be integrated into A3ME framework to extend their functionality and to reuse the existing solutions in the A3ME framework.

3.5.1 Jini / Apache River

Jini is a service oriented architecture based on Java technology which allows to build secure, distributed systems of computers and devices running Java in a local IP network. The devices can offer, discover and use services of each other.

The Jini architecture is designed to deploy and to use services in a dynamic network, where things are added, removed, changed and parts of the network can fail and be repaired again. To allow this Jini focuses on a few simple principles [47]:

- Remote objects,
- Leasing (commitments in a Jini system are of limited duration),
- Distributed events (events aren't as predictable as on a singe machine),
- Two-phase commit (because network can fail).

Originally Jini [3] started as Sun project. But since it was not as successful as expected and to attract new developers Sun changed the license from SCSL (Sun Community Source License) to an open source license (Apache License, Version 2.0). In 2006 it was handed over from Sun to Apache and in 2008 it started incubating as Apache River project[10]. Apache River project's latest release was released as version 2.2.2 on November 18, 2013.

The basic communication between client and service is based on RPC (Remote Procedure Call). Client and service communicate with a protocol called JERI. There are JERI implementations for plain-TCP, plain-SSL, HTTP, HTTPS and Kerberos-TCP. For compatibility with RMI (Remote Method Invocation) there is also a JRMP (Java Remote Method Protocol) transport. The smallest Jini/river system uses only JERI and is comprised of a service and a client. [151]

Jini/river uses a lookup service for registration and discovery of services. Once a service is found a proxy object is used to interact with the remote object which implements the service. The services have a limited validity and their lease needs to be renewed periodically.

The use of a central lookup makes Jini a centralized solution. This makes it less scalable and also introduces a single point for failures. Another weakness is that all devices require a Java VM to run Jini. This means Jini/river can not run on resource constrained devices like TelosB (see section 5.6.1).

Jini is well suited to offer services between Java enabled devices in a local IP network, which makes it a valuable extension possibility of the A3ME framework in the future work.

3.5.2 CORBA

The Common Object Request Broker Architecture (CORBA) is a middleware which enables language and a platform-neutral remote procedure call (RPC). It is defined as a set of standards: CORBA Interfaces [26], CORBA Interoperability [27] and CORBA Component Model [28] by the Object Management Group (OMG)[11] that enables software components written in different computer languages and runs on multiple devices to work together.

An Interface Definition Language (IDL) is used to define methods together with their parameters independent of the programming language. It also provides the information needed to develop clients that use the interface's operations. This allows coupling of components implemented in different programming languages. IDL is a declarative language described in Extended Backus-Naur Form (EBNF). The IDL is used to describe the interfaces. The implementations of it are mapped to different programming languages. The different mappings are defined in the specification. [26]

[10] Project web site: http://river.apache.org.

[11] OMG web page: http://www.omg.org

All interactions among applications and objects are initiated through Object Request Broker (ORB), which must be initiated on each participating device. The ORBs communicate with each other using General InterORB Protocol (GIOP). The applications don't have to deal with communication issues when interacting with remote objects. The encoding, transmission and decoding is handled by ORB. This considerably reduces the complexity of distributed applications.

CORBA is designed for use on conventional computers and servers. Lightweight embedded variations of the traditional CORBA service are also being developed for accommodating resource constrained devices [19]. "However, CORBA is inherently based on request/response synchronous communication model, which is a misfit for the nature of WSNs, where the communication is packet based, highly variable in speed, and error-prone." [116]

CORBA is an established well designed middleware covering the communication aspects for distributed computing and abstracting from the hardware and programming heterogeneity of the components.

This work has a different focus, namely enabling ad-hoc discovery and interactions of devices and their capabilities in a decentralized way. This positions our solutions orthogonal to CORBA – covering a different problem space. Our solutions could be combined with CORBA to offer the benefits from both. A3ME could use the CORBA services and communication mechanisms as soon as an ORB is available among known devices in communication range. And an ORB could use A3ME mechanisms to offer device and capabilities discovery functionality.

3.5.3 Web Services

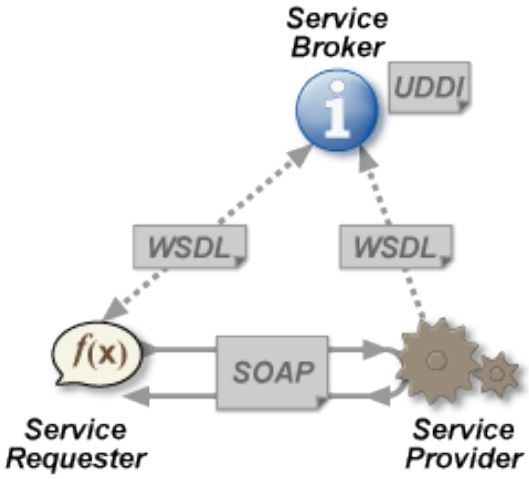


Figure 9: Web Services Architecture (Source: en.wikipedia.org/wiki/Web_service. Licensed under GNU Free Documentation License).

Web Services (WSs) is the standard way of offering and using services in the World Wide Web (WWW). A Web Service is described using Web Services Description Language (WSDL) based on XML. WSDL contains the implementation independent description of the offered service. It is usually published by the service provider and can also be registered by a Service Broker, where service consumer can search for services. [50]

There can be more than one service provider for a WS described by a WSDL. A WS consumer can generate a the program for any of the many supported programming languages directly from the WSDL description file and use it to call the service.

By design the WSs are supposed to be searched/looked-up on Service Brokers, but in practice most of these services are known a-priori and not looked up on a broker as shown in figure 9.

To build or use Web Services many tools and tool chains exist. This simplifies the development and maintenance of WS-based software. A variety of additional techniques extend the WSs with further functionality:

- various authentication techniques,
- various encryption techniques,
- serialization techniques,
- etc.

WSs are based on TCP/IP and HTTP protocols which also bring their resource requirements which are not met by many resource constrained devices.

3.5.4 JXTA

JXTA (Juxtapose)[12] [13] is a programming language and platform independent Open Source protocol started by Sun Microsystems for peer-to-peer (P2P) networking in 2001. It was later continued as the Kenai[14] project and is now being moved to java.net[15]. JXTA technology can be used to create peer-to-peer (P2P) applications. It is a set of open protocols that enables connected devices on the network to communicate and collaborate in a P2P manner. JXTA peers create a virtual network where any peer can interact with other peers and resources directly. [136]

The JXTA protocols are defined as a set of XML messages, which allow any device, connected to a network to exchange messages and collaborate independently of the underlying implementation. The JXTA protocols standardize the manner in which peers [18]:

- Discover each other,
- Self-organize into peer groups,
- Advertise and discover network resources,
- Communicate with each other,
- Monitor each other.

There are three major implementations of JXTA:

- Jxta-jxse: JXTA for Java 5.0 SE/EE,
- Jxta-jxme: JXTA for Java Micro Edition,
- Jxta-c: C/C++/C# implementation of JXTA.

The IETF declined the assignment of the JXTA to a working group in 2002. In November 2010, Oracle officially announced its withdrawal from the JXTA projects. The community voted to move the project to the Apache Software Foundation (ASF), but after Oracle announced that it would not transfer the JXTA trade name the transition was frozen. Now the projects are being moved from Kenai to the java.net platform.

The use of XML messages reduces the possibility to use JXTA on resource constrained devices, what is one of the requirements of this work.

3.5.5 UPnP

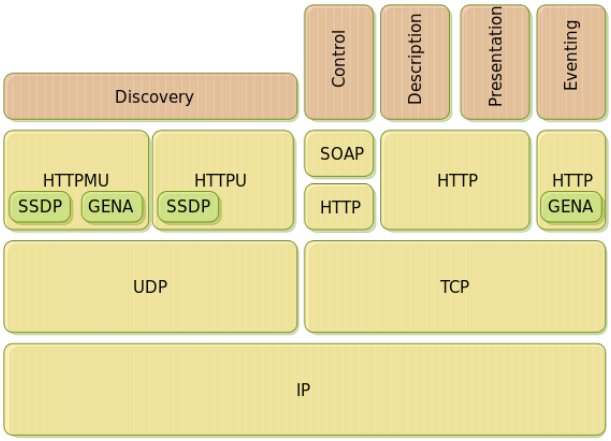


Figure 10: UPnP Architecture (Source: http://de.wikipedia.org/wiki/Universal_Plug_and_Play)

The Universal Plug and Play (UPnP)[16] is a framework for (self-)configuration and control of devices in an IP-based network. It defines protocols for device discovery and description. UPnP is usually used by multimedia devices and personal computers in local IP networks.

The UPnP architecture (Figure 10) enables automatic self-configuration of devices in an IP network. Any UPnP compatible device can dynamically join the network and announce its name and capabilities on request. It can also discover other devices and their capabilities. [65]

The UPnP Device Architecture Version 1.1 [65] and 21 standard UPnP Device Control Protocol specifications (Table 1) were adopted and published by the International Standards Organization (ISO) and International Electrotechnical Commission (IEC) as International Standards in the fall of 2011. In February 2015 the UPnP version 2.0 was published

[12] Project JXTA Website on Kenai (previous project hosting): http://jxta.kenai.com/.
[13] Project JXTA Website on java.net (new project hosting): http://java.net/projects/jxta/.
[14] Kenai is a collaborative hosting site for free and open source projects, launched by Sun Microsystems and now owned by Oracle.
[15] Java.net is a community of Java developers and their projects hosted by Oracle.
[16] UPnP forum: http://upnp.org.

Audio/Video – MediaServer:4 and MediaRenderer:3 – MediaServer:3 – MediaServer:2 and MediaRenderer:2 – MediaServer:1 and MediaRenderer:1
Basic – Basic Device:1
Device Management – ManageableDevice:1 – ManageableDevice:2
Home Automation – SolarProtectionBlind:1 – Digital Security Camera:1 – HVAC:1 – Lighting Controls:1
Networking – Internet Gateway:2 – Internet Gateway:1 – WLAN Access Point:1
Printer – Printer Enhanced:1 – Printer Basic:1
Remote Access – RAServer:2 and RADiscoveryAgent:2 – RAClient:1, RAServer:1 and RADiscoveryAgent:1
Remoting – Remote UI Client:1 and Remote UI Server:1
Scanner – Scanner:1
Telephony – Telephony:1

Table 1: UPnP Standard Device Control Protocols (SDCPs)

[66]. Here among others the IPv6 was made mandatory: a device and a control point now shall support dual stack (IPv4 and IPv6) operation to be certified.

All communication in UPnP is based on top of Internet Protocol (IP). IPv6 can also be supported but is not required on UPnP devices. For discovery the UDP and for everything else TCP is used. The devices can exchange device and service descriptions in XML format. The messages are exchanged with the SOAP (section 3.8.3) protocol.

The DeviceProtection service [114] enables Devices to provide privacy and restrict access to sensitive operations to authorized devices and users. The secure communication is performed through the Transport Layer Security (TLS) protocol [64]. If secure communication is used, devices must use X.509 certificates to authenticate themselves otherwise only the publicly available functionality can be used.

The UPnP E-Health and Sensors (EH&S) Working Committee intends to develop standards to address the management of sensor networks, ecosystem specific data aggregation and messaging between devices. [118]

The UPnP Forum continues to develop specifications for new scenarios and new types of applications. The main difference to our framework is the limitation to IP based communication. For IP capable devices in A3ME it is reasonable to reuse the UPnP functionality and offer it also to other A3ME devices and at the same time to offer the information available through A3ME via UPnP (section 5.3.5).

The listing 3 shows an example of a simple device description. Since the description is done using XML, the size of the resulting message is relatively high. For some resource constrained devices this is already a criterion for exclusion.

Many consumer devices already support UPnP and provide their descriptions to others making it an optimal choice to be reused in the A3ME to discover devices, which do not support A3ME directly, and get their capabilities.

```
<?xml version="1.0"?>
<root xmlns="urn:schemas-upnp-org:device-1-0">
    <specVersion>
        <major>1</major>
        <minor>0</minor>
    </specVersion>
    <device>
        <deviceType>urn:schemas-upnp-org:device:BinaryLight:1</deviceType>
        <friendlyName>UPnP Binary Light</friendlyName>
        <manufacturer>MyCompany</manufacturer>
        <manufacturerURL>www.mywebsite.org</manufacturerURL>
        <modelDescription>New brilliant BinaryLight</modelDescription>
        <modelName>SuperWhiteLight 4000</modelName>
        <modelNumber>1</modelNumber>
        <UDN>uuid:138d3934-4202-45d7-bf35-8b50b0208139</UDN>
        <serviceList>
            <service>
                <serviceType>urn:schemas-upnp-org:service:SwitchPower:1</serviceType>
                <serviceId>urn:upnp-org:serviceId:SwitchPower:1</serviceId>
                <SCPDURL>switchpower_scpd.xml</SCPDURL>
                <controlURL>/control</controlURL>
                <eventSubURL>/eventing</eventSubURL>
            </service>
        </serviceList>
    </device>
</root>
```

Listing 3: UPnP Device Description Example[89]

3.5.6 FIPA

Foundation for Intelligent Physical Agents FIPA defines a set of standards for multi-agent based systems. Among others FIPA defines:

- FIPA Abstract Architecture Specification [7],
- FIPA ACL Message Structure Specification [9],
- FIPA Communicative Acts (Table 2) [10],
- FIPA Interaction Protocols [4].

The communication between agents in FIPA is message based. FIPA communicative acts [10] are based on the speech act theory: which says that each message implies some type of action. FIPA defines 22 different communicative acts – performatives (Table 2).

FIPA standards themselves are mainly used in the area of multi-agent systems (MAS). They also have influenced many other technologies like web services, internet of things, etc. Table 3 shows a list of systems, which use FIPA standards. The most known of them is the JADE framework (see section 3.5.7) developed by Telecom Italia.

In A3ME we use FIPA performatives as message types and also took over the message parameters defined in the FIPA ACL Message Structure Specification.

3.5.7 JADE

JADE[18] (Java Agent DEvelopment Framework) is an open source software framework which allows to implement multi-agent based software. JADE platform can be distributed over multiple machines. JADE agents are software programs capable (but not required) to move between execution platforms together with their execution state. Each agent usually is specialized on one task and interacts with other agents to achieve its task. JADE is compliant with the FIPA (section 3.5.6) specifications. It requires Java Runtime Environment 1.4 or higher.

An application based on JADE is made of a set of components called Agents (figure 11) each one having a unique name. Agents execute tasks and interact by exchanging messages. Each agent is assigned and registered to a platform which provides the basic services like message delivery. A platform is composed of one or more Containers, which can

[17] JIAC web page: http://www.jiac.de/.

[18] JADE Java Agent DEvelopment Framework web page http://jade.tilab.com.

Speech act	Description
Accept Proposal	The action of accepting a previously submitted proposal to perform an action.
Agree	The action of agreeing to perform some action, possibly in the future.
Cancel	The action of one agent informing another agent that the first agent no longer has the intention that the second agent performs some action.
Call for Proposal	The action of calling for proposals to perform a given action.
Confirm	The sender informs the receiver that a given proposition is true, where the receiver is known to be uncertain about the proposition.
Disconfirm	The sender informs the receiver that a given proposition is false, where the receiver is known to believe, or believe it likely that, the proposition is true.
Failure	The action of telling another agent that an action was attempted but the attempt failed.
Inform	The sender informs the receiver that a given proposition is true.
Inform If	A macro action for the agent of the action to inform the recipient whether or not a proposition is true.
Inform Ref	A macro action for sender to inform the receiver the object which corresponds to a descriptor, for example, a name.
Not Understood	The sender of the act (for example, i) informs the receiver (for example, j) that it perceived that j performed some action, but that i did not understand what j just did. A particular common case is that i tells j that i did not understand the message that j has just sent to i.
Propagate	The sender intends that the receiver treat the embedded message as sent directly to the receiver, and wants the receiver to identify the agents denoted by the given descriptor and send the received propagate message to them.
Propose	The action of submitting a proposal to perform a certain action, given certain preconditions.
Proxy	The sender wants the receiver to select target agents denoted by a given description and to send an embedded message to them.
Query If	The action of asking another agent whether or not a given proposition is true.
Query Ref	The action of asking another agent for the object referred to by a referential expression.
Refuse	The action of refusing to perform a given action, and explaining the reason for the refusal.
Reject Proposal	The action of rejecting a proposal to perform some action during a negotiation.
Request	The sender requests the receiver to perform some action. One important class of uses of the request act is to request the receiver to perform another communicative act.
Request When	The sender wants the receiver to perform some action when some given proposition becomes true.
Request Whenever	The sender wants the receiver to perform some action as soon as some proposition becomes true and thereafter each time the proposition becomes true again.
Subscribe	The act of requesting a persistent intention to notify the sender of the value of a reference, and to notify again whenever the object identified by the reference changes.

Table 2: FIPA-ACL Performatives

Jade (see section 3.5.7)
Java-based Intelligent Agent Componentware (JIAC) [17]
The SPADE Multiagent and Organizations Platform (Python)
The Spyse agent platform (Python)
JACK Intelligent Agents (Java)
The April Agent Platform (AAP) and Language (April) (No longer actively developed)
The Fipa-OS agent platform (No longer actively developed)

Table 3: Systems using FIPA Standards

be running on different hosts. Each container can contain multiple agents. One of the Containers must be the Main container providing central platform services and all other containers must register on it. [150]

Agents communication is based on an asynchronous message passing paradigm. Message format is defined by the ACL language defined by FIPA (see section 3.5.6). The agents are addressed via Agent IDs (AIDs) composed of local and platform name: <*local-name*>@<*platform-name*>. The AID can contain one or multiple transport addresses like IPs.

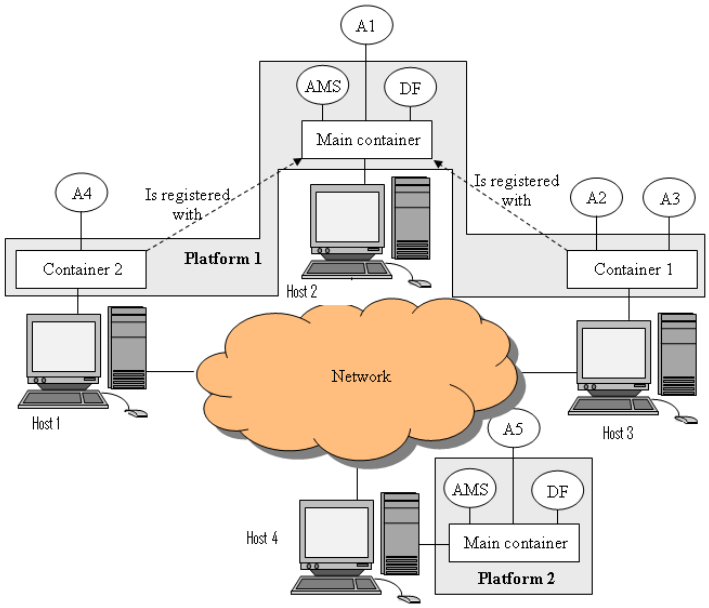


Figure 11: The JADE Architecture [150]

In Jade an agent encapsulates a software task composed of program code and state. In contrast in A3ME a device-agent represents a physical device and offers its information and services. The idea of separating the ID and the communication addresses for our framework was inspired by JADE's AIDs.

3.5.8 Lightweight Publish/Subscribe

In many scenarios in MME there are decoupled producers and consumers of information. Publish/subscribe (pub/sub) communication paradigm offers the mechanisms for publishing, storage and distribution of the information in such scenarios. Most conventional pub/sub mechanisms are not convenient for resource-constrained devices, but some recent solutions are designed specifically for such devices.

The μC-SemPS [134] is a lightweight semantic pub/sub system which focuses on the energy efficiency to extend the network lifetime. To achieve this the matching and routing of the semantic events is optimized to reduce computation and communication costs. μC-SemPS assumes a homogeneous communication technology among the nodes and is therefore not applicable to heterogeneous networks.

The Message Queue Telemetry Transport (MQTT) [23] is a lightweight topic-based pub/sub messaging protocol over TCP/IP designed for machine-to-machine and mobile applications. It supports three Quality of Service (QoS) levels:

- "At most once",
- "At least once" and
- "Exactly once".

Since this protocol is build on top of TCP/IP it is not directly applicable to heterogeneous networks.

The extended and modified version of MQTT For Sensor Networks (MQTT-SN) [146] is developed for heterogeneous networks. It is adapted to the peculiarities of a wireless communication environment and of battery-powered resource-constrained devices.

We consider MQTT-SN to be the right protocol to add on top of A3ME to offer pub/sub communication paradigm. As pub/sub topics the A3ME classification can be used, which would reduce the size of the MQTT-SN messages.

3.5.9 CoAP

The Constrained Application Protocol (CoAP) is a protocol aligned to HTTP but specialized for resource constrained devices and unreliable communication. It uses client/server approach, but each device can usually acts in both client and server roles. Clients send requests for an action on a resource on the server and get a response with a code and a resource representation if requested. The messages are transported asynchronously using datagram-oriented transport protocols. CoAP defines four types of messages: Confirmable, Non-confirmable, Acknowledgement, Reset. Reliable delivery of messages can be assured by marking a message as Confirmable. A Confirmable message is retransmitted using a timeout and exponential back-off. [144]

CoAP offers proxy functionality, which might be used in WSNs to answer requests on behalf of sensors by intermediate nodes if the answer is still valid to save energy.

CoAP is the equivalent of HTTP for resource constrained devices and communication. It offers the basic request/response mechanisms which can be reused for different tasks. A3ME offers these functionality itself and allows more compact messages. It would be possible to replace the message delivery in A3ME with CoAP at the cost of having additional 4 bytes of fixed-length CoAP header.

3.6 Neighbor Discovery

With the rise of the number of mobile and embedded devices with wireless communication capabilities in our environment the device discovery becomes crucial to enable the interactions between these devices. Neighbor discovery is the process of acquiring information about other devices in the vicinity of a device. Hereby the following information is acquired [63]:

- presence of other devices,
- capabilities of other devices, and
- information available at other devices.

Usually this discovery is done inside of a specific communication network. But there is also research going on focusing on inter-network solutions. The ACROPOLIS Network of Excellence (NoE) investigates neighbor and network discovery in cognitive radio networks [63] along with the inter-network communication. Even inside of a specific network there are usually multiple communication channels available for communication. So the challenge is to use a common channel for the discovering and the listening node at some time interval.

3.6.1 Neighbor Discovery in Multi-channel Networks

Widely spread wireless communication technologies based on the IEEE 802.11 [34] are wireless local area networks (WLANs) and mobile ad hoc networks (MANETs). In typical WLANs the access points are stationary and the client devices only have to discover the access points to connect to. In MANETs all the devices are mobile and have to rediscover the neighborhood periodically to maintain the network connectivity. In IEEE 802.11 multiple channels can be used for communication. This means that any two devices need to use a common channel at some point in time to be able to discover each other and to communicate.

In [101] the quorum system is applied to the selection of communication channels for the discovery of neighbors. A *quorum system* is a collection of pairwise non-empty subsets of elements of a common set with n elements [56] [153]. There are different possibilities to build a valid quorum system: majority, cyclic, grid [106], etc. The *quorum-based* neighbor discovery guaranties that after n time intervals two nodes will discover each other.

MANET Neighborhood Discovery Protocol (**NHDP**, RFC 6130) [59] describes a 1-hop and symmetric 2-hop neighborhood discovery. In NHDP each node uses 1-hop HELLO messages to advertise itself. These HELLO messages are send out periodically. The repetition period can be static or dynamic. A HELLO message is also send when either the knowledge about the node itself or about its 1-hop neighbors changes. The HELLO messages can contain the information about the nodes network addresses, neighbors and whether the links are bi-directional allowing to collect 2-hop information. The collected information is stored locally and can be reused by other protocols.

In [87] an adaptive HELLO messaging scheme for neighbor discovery in on-demand MANET routing protocols is proposed. The interval for sending HELLO messages here is adjusted to the average interval for sent or received messages of the node. This allows to avoid sending unnecessary HELLO messages, without significantly increasing the risk to miss a broken link to a neighbor.

Bluetooth uses frequency hopping on 79 channels. Bluetooth devices are only discoverable if they are in the discoverable mode – meaning they are listening for discovery messages. The Bluetooth range depends on the Bluetooth power class of the device and is accordingly about 1, 10 or 100 meters. While discovering a Bluetooth device repeatedly sends *inquiry* messages while hopping between different frequency channels. Other devices in discoverable mode listen periodically for inquiry messages on different channels. Once they receive a *inquiry* message, they can (but don't have to) answer to it. The inquiry can be specified to be for all Bluetooth devices or only for specific types of devices. [37]

In a project for cloud density estimation [157] Bluetooth device discovery was used to estimate the number of people per square meter. Hereby the native Bluetooth discovery was used with a scan interval of 60 seconds.

In **Bluetooth Low Energy (BLE)** networks only 3 of the 40 channels are assigned to advertising and tiny frames are used. An advertising is send consecutively on all three advertisement channels and is repeated after the repetition time which can be between 20ms and 10.24s. In [112] a mechanism is proposed to automatically tune the parameters of BLE to optimize the latency in crowded BLE networks.

3.6.2 Neighbor Discovery in Single-channel Networks with Low Duty Cycles

In WSNs usually all nodes of a specific WSN use a common communication network and channel making the search for the right communication network and channel unnecessary. But what still makes the neighbor discovery difficult here, is

the fact that most protocols in WSN switch off the radio of the nodes periodically to safe energy. To discover each other the radio components of the sending and receiving nodes have to be on at the same time. To manage the different states the time is usually divided into time slots. Two nodes then must have an active slot in common which overlaps long enough to enable communication.

A survey about neighbor discovery in WSNs [77] identified three basic methods to enable neighbor discovery:

- Randomness to select when to be awake and when to sleep,
- Special patterns of awake slots, which guarantee a common active slot, or
- A node must remain awake for multiple slots to ensure discovery.

In most neighbor discovery protocols either one of these methods or a combination of those is used.

Probability based protocols like the **Birthday** protocol [117] use the randomness to decide in each time slot whether the node shall sleep, send or listen. The time required for detection (with specified probability) and the sleep/awake ratio is controlled by the probabilities for the three states.

Deterministic methods ensure the discovery after n slots. These methods are usually based on quorum systems. A quorum system is a collection of pairwise non-empty subsets of elements of a common set with n elements (see also section 3.6.1). The quorum set defines in which slots a node has to be active. The *Brute Force* method corresponds to a majority-quorum, where a node stays active for more than 50% of the slots. An *enhanced Brute Force* Method described in [77] corresponds to the grid-quorum and allows a node to stay asleep for more than 50% of the time, but increases the time after which the neighbor discovery is guaranteed. The "**Quorum**" method [153] often used in literature related to neighbor discovery uses a grid-quorum system, where the slots are arranged in a square and each node chooses a row and a column of slots in which it stays awake.

Another method to build a quorum system is using *prime numbers*. Here the length of a round after which an awake slot comes is a prime number. If all neighbor nodes use different primes for their round length the discovery is guaranteed. The quorum size, which corresponds to the worst case discovery time, is then the product of the primes of the two neighbors. The problem with these prime number based methods is to ensure that all neighbors use different primes. To overcome this problem a combination of primes for each node can be used. The node stays awake when the round number is divisible by any of the node's primes. This technique also has the advantage that the duty cycle can be adjusted at a finer grain. The duty cycle in this case can be calculated as sum of reciprocals of the node's primes. The **Disco** protocol [71] chooses a pairs of primes to get a desired duty cycle.

U-Connect [100] protocol is a combination of a prime number based and a majority-quorum methods. The nodes are in listening mode in every slots divisible by p, where p is a prime number. And at every p^2 slot each node transmits for the duration of $\frac{p}{2}+1$ slots. This protocol is proved in [100] to be a 1.5-approximation to the optimal solution and outperforms the Disco and grid-quorum methods when using the power-latency metric defined in [100].

Li at al. [111] proposed a dynamic adjustment of the duty cycles to accelerate the adaptation to the network changes in a mobile WSN by predicting the number of new nodes needed to be discovered.

Searchlight [48] combines the probabilistic and the deterministic methods. It offers average case discovery latency comparable to the probabilistic methods while offering best worst case latency. The algorithm uses two active slots: anchor and probe slot in each period of t slots. The anchor slot has always the position 0 a the beginning of a period and the probing slot gets the positions $1, \ldots, \lfloor \frac{t}{2} \rfloor$. To improve the intersection of the probing slots with each other the pattern for the position of the probing slot across the periods is randomized for each node. This further improves the average case discovery.

3.6.3 Neighbor Discovery in Multi-Channel Network with Low Duty Cycles

In [161] **McDisc** is introduced which describes a reliable neighbor discovery protocol in low duty cycle and multi-channel wireless networks. For each channel it uses the U-Connect protocol (section 3.6.2). And for neighbor discovery across multiple channels it introduces two approaches: a probabilistic (McDisc-R) and a deterministic one (McDisc-D). In McDisc-R for each active slot the channel is used randomly. This results in a good average discovery ratio but unbound worst-case discovery latency. In McDisc-D the channel is calculated by *slot-number* modulo *channel-count*. This method offer the same characteristics as U-connect for single channel networks, but has a draw back of working only if the slots are synchronized.

3.6.4 Neighbor Discovery on Network Layer

On the network layer it is not required to deal with the channels and the duty cycles, because those are considered to be dealt on the lower layers of the OSI model. IPv6 neighbor discovery optimizations for wired and wireless networks [55] is an IETF internet-draft, which is currently under review. It incorporates techniques introduced in neighbor discovery optimization for 6LoWPANs (IPv6 over Low power Wireless Personal Area Networks) (RFC6775) [54] and will update RFC4861 [126] when approved. According to this protocol a device sends "Router Solicitation" as a multicast message when it is powered on and receives "Router Advertisements" as answers from the routers. The information in the answers

is used to form an IPv6 address which is then send to address registrars, whose addresses were included in the router advertisement, to register it. If the registration succeeds the registrars send a neighbor advertisement message. The addresses have an expiration timestamp and have to be refreshed by each device before they expire.

3.7 Service Discovery

The service discovery in general can be done using a central instance, a set of brokers or by asking each device directly for its services.

Different middleware solutions offer their own centralized service directories for registration, management and search of services. Universal Description, Discovery and Integration (UDDI) [53] was supposed to serve as a central look up for Web Services and their realizations by different parties, but was not widely used. Java Naming and Directory Interface (JNDI) [129] is a Java API to access different directory and naming services in Java.

In Universal Plug and Play (UPnP, section 3.5.5) the devices can register themselves at a control point or a control point can search for devices using Simple Service Discovery Protocol (SSDP). SSPD is a text based protocol used in local IP networks. It uses a predefined multicast IP address to for advertisements and discovery of services using UDP/IP protocol.

The Efficient semAntic Service discoverY (EASY) [122] is a service discovery protocol which can be build on top of other existing service discovery protocols (SDPs). It defines a language for semantic specification of functional and non-functional service properties and a matching mechanism for the different underlying SDPs together with a rating system for the services. This solution is not designed for resource-constrained devices and can be therefore only used on unconstrained devices.

AIDAS [152] offers a user-centric semantic-based service discovery and filtering to provide personalized views on the available services to the users. The framework uses XML messages and is therefore only usable on rich clients and not on resource-constrained devices.

CoAP (section 3.5.9) has a build in functionality to ask each individual device for it's offered services. CoAP's proxies can be reused as brokers for service discoveries.

3.8 Generic Data Definition and Serialization/Deserialization Technologies

Even today in many projects proprietary data serializations (encodings) for messages, queries, etc. are used. It is especially the case in the areas like in WSNs, WSANs, Ubiquitous Environments and robots where resource constrained devices are used. With this work we would like to change it and demonstrate that it is more convenient to use existing standards.

In the first place to use a standard means to invest time to learn what standard to use and how. In many cases also money must be spent for tools and/or licenses. On the other hand time for developing grammar, program structures, parsers, encoder and decoders can be saved. Additionally it allows to get generated code for more than one programming language. This means in the end to get a better product and to save money.

3.8.1 XML

Extensible Markup Language (XML) describes a class of data objects called XML documents and partially describes the behavior of computer programs which process them. XML is an application profile or restricted form of SGML, the Standard Generalized Markup Language [1]. By construction, XML documents are conforming SGML documents. [20]

XML documents are widely used in world wide web related technologies, for example in:

- Semantic Web,
- Web Services,
- SOAP,
- etc.

XML documents can be assigned to a XML schema allowing to check if the document is compliant to it. Cascading Style Sheets (CSS) can be used to specify the presentation layout of the XML documents.

In A3ME we decided not to use XML documents because of their relatively big size relative to information content and because of the high computing and memory requirements to parse XML documents. Nevertheless since we are using ASN.1 based data definitions it is possible to encode those to XML documents with the XML Encoding Rules (XER) (see section 3.8.5.2).

3.8.2 Efficient XML Interchange (EXI) Format

Efficient XML Interchange (EXI) Format [39] is a representation format for XML documents developed by W3C's Efficient XML Interchange Working Group. It can be used for any XML documents (schema-less) and for schema-informed documents based on a specific XML schema definition (XSD). EXI representation of the XML documents significantly reduces

the size of the documents, especially for schema-informed documents, compared to the usual plain text representation of XML documents. The compression can further be improved by using optional EXI Compression. Advantage of using EXI is the possibility of direct access and modification of the content without decoding it back to plain text representation [21].

In A3ME we decided to use ASN.1 for content definition and representation in combination with ASN.1 PER encoding, while XML content in combination with XSD definitions and EXI representation/encoding would be the next reasonable choice.

3.8.3 SOAP

SOAP is a protocol used to exchange structured information in a decentralized, distributed environment e.g. in Web Service implementations. The messages are defined in XML and can be exchanged over a variety of underlying protocols. The framework has been designed to be independent of any particular programming model and other implementation specific semantics. Two major design goals for SOAP are simplicity and extensibility. [81]

The acronym SOAP originally stood for 'Simple Object Access Protocol' but was dropped with Version 1.2 of the standard.

The SOAP Version 1.2 specification consists of three parts:

- Part 0: Primer [121],
- Part 1: SOAP messaging framework [81],
- Part 2: Adjuncts [82].

The SOAP messaging framework consisting of:

- The SOAP processing model defining the rules for processing a SOAP message.
- The SOAP Extensibility model defining the concepts of SOAP features and SOAP modules.
- The SOAP underlying protocol binding framework describing the rules for defining a binding to an underlying protocol that can be used for exchanging SOAP messages between SOAP nodes.
- The SOAP message construct defining the structure of a SOAP message.

The example in listing 4 shows an example soap message.

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
 <env:Header>
  <n:alertcontrol xmlns:n="http://example.org/alertcontrol">
   <n:priority>1</n:priority>
   <n:expires>2014-09-22T14:00:00-05:00</n:expires>
  </n:alertcontrol>
 </env:Header>
 <env:Body>
  <m:alert xmlns:m="http://example.org/alert">
   <m:msg>Pick up Mary at school at 2pm</m:msg>
  </m:alert>
 </env:Body>
</env:Envelope>
```

Listing 4: SOAP message containing a SOAP header block and a SOAP body [81]

The SOAP protocol is well structured and extensible to fit different scenarios and needs. The disadvantage is that it is not human friendly even so it might be used in combination with special style sheets to make it easier to read for humans. Another two disadvantages come up when we consider to use SOAP on resource constrained devices. First since SOAP is based on XML the participating devices must be able to generate and to parse XML content. This might be a bottleneck in terms of required computing power, program space and memory. The second disadvantage is the size of the resulting message. Since the XML tags and definitions add up to the size of the message, the resulting payload which needs to be transmitted is much bigger than the size of the actual usable information. Considering that communication is the most energy consuming activity on wireless sensors/actors, it is a reason to check for another solution in this area.

3.8.4 JSON - JavaScript Object Notation

JSON (JavaScript Object Notation) is a lightweight data-interchange format based on a subset of the JavaScript programming language. It is easily readable for humans and is simple for machines to parse and generate it. JSON is a text format that is language independent but uses conventions that are familiar to programmers of popular programming languages like C and Java. [40] [38]

This serialization/deserialization technology has the advantage of not requiring any sophisticated libraries to use it, but it also is not byte length efficient.

```
{
     "firstName": "Arthur",
     "lastName" : "Herzog",
     "age"      : 36,
     "address"  :
     {
         "streetAddress": "Hochschulstrasse 10",
         "city"         : "Darmstadt",
         "postalCode"   : "64289"
     }
 }
```

Listing 5: JSON example

In A3ME we decided not to use JSON because the resulting message size is not optimal and the messages can not be validated to contain some specific data structure as it is possible for XML and ASN.1.

3.8.5 ASN.1

Abstract Syntax Notation One (ASN.1) is a joint ISO/IEC and ITU-T standard, originally defined in 1984. It was revised in 1995 and the latest available version is dated 2002. This standard is used for the definition of messaged of many established protocols for [97]:

- Lightweight Directory Access Protocol (LDAP),
- Security, authentication, and cryptography,
- Biometrics,
- Banking,
- Mobile telephony and wireless networks,
- Wired telephone networks,
- etc.

In addition to the ASN.1 a set of encoding rules is defined to encode and decode the data defined in ASN.1 into different formats.

We identified ASN.1 in combination with ASN.1 Packed Encoding Rules (PER, section 3.8.5.2) as the best fitting solution for A3ME. ASN.1 offers a programming language independent data type definition, allowing to define very complex data with optional and extendable elements. The data based on this definition can be encoded in byte length efficient way by use of the ASN.1 PER, since the definition elements don't have to be included, if sender and receiver knows the definition.

3.8.5.1 ASN.1 Definitions

ASN.1 is a widely established standard to specify data structures independent of the device and communication technology used. Listing 6 shows an example definition of contact data and Listing 7 shows a corresponding sample data definition. The typical application area for ASN.1 is the definition of protocol messages. [158]

```
{
Contact ::= SEQUENCE {
  firstName IA5String,
  lastName  IA5String,
  age INTEGER,
  address SEQUENCE {
    streetAddress IA5String,
    city  IA5String,
    postalCode INTEGER
  }
}
```

Listing 6: ASN.1 example definition of the contact data structure.

```
{
contact Contact ::={
  firstName: "Arthur",
  lastName : "Herzog",
  age      : 35,
  address
  {
    streetAddress: "Hochschulstrasse 10",
    city         : "Darmstadt",
    postalCode   : 64289
  }
}
```

Listing 7: ASN.1 example data for the contact structure defined in listing 6

3.8.5.2 ASN.1 Encoding rules

ASN.1 encoding rules are sets of rules used to encode/decode data described in ASN.1 language. A given ASN.1 definition can be encoded/decoded by any ASN.1 encoding rule to and from a byte array or into an XML document when using Extended XML Encoding Rules.

As the structure of ASN.1 (see section 3.8.5) is hierarchical, the basic encoding rules follow this structure. They operate on identifier, length, contents scheme (ILC). The structure is therefore recursive such that the contents can be a series of ILCs. This bottoms out with basic data types such as a text string or an integer (see table 4).

Universal Tag Number	Description
0	reserved for BER
1	BOOLEAN
2	INTEGER
3	BIT STRING
4	OCTET STRING
5	NULL
6	OBJECT IDENTIFIER
7	ObjectDescriptor
8	INSTANCE OF, EXTERNAL
9	REAL
10	ENUMERATED
11	EMBEDDED PDV
12	UTF8String
13	RELATIVE-OID
16	SEQUENCE, SEQUENCE OF
17	SET, SET OF
18	NumericString
19	PrintableString
20	TeletexString, T61String
21	VideotexString
22	IA5String
23	UTCTime
24	GeneralizedTime
25	GraphicString
26	VisibleString, ISO646String
27	GeneralString
28	UniversalString
29	CHARACTER STRING
30	BMPString

Table 4: Universal Tags in ASN.1

The ASN.1 encoding rules currently standardized are [130]: Basic Encoding Rules (BER), Distinguished Encoding Rules (DER), Canonical Encoding Rules (CER), Packed Encoding Rules (PER), XML Encoding Rules (XER) and Extended XML Encoding Rules (E-XER).

BER: The Basic Encoding Rules (BER) are historically the original encoding rules of ASN.1 since they were already part of the [X.409] standard before it was split up into two parts in 1985. The term 'basic' indicated that other rules might be standardized in the future; it actually happened in 1994 when the packed encoding rules (PER) were introduced in the standard. [69, page 394]

DER: The Distinguished Encoding Rules (DER) is a specialized form of BER that is used in security-conscious applications. These applications, such as electronic commerce, typically involve cryptography, and require that there be one and only one way to encode and decode a message. [130]

CER: Canonical Encoding Rules (CER) is another specialized form of BER that is similar to DER, but is meant for use with messages so huge that it is easiest to start encoding them before their entire value is fully available. CER is rarely used, as the industry has locked onto DER as the preferred means of encoding values for use in secure exchanges. [130]

PER: Packed Encoding Rules (PER) is more recent than the above sets of encoding rules and is noted for its efficient algorithms that result in faster and more compact encodings than BER. PER is used in applications that are bandwidth or CPU starved, such as air traffic control and audiovisual telecommunications. [130]

XER: XML Encoding Rules (XER) allow to encode a message that has been defined via ASN.1 using XML. XER allows to visualize the ASN.1-described messages via XML. [130]

E-XER: Extended XML Encoding Rules (E-XER) is an amendment to the ITU-T Rec. X.693 (23002) ASN.1 Encoding Rules: Specification of XML Encoding Rules (XER). Extended-XER encoding makes ASN.1 an XML schema notation as powerful as XSD, with the simplicity of ASN.1. [130]

Tool name	Notes	for Java	for c	PER	free
OpenASN (Diploma thesis at TUD) [95]	open source, does not support UTF8String data type, uses parameterized types, work only for Java version 1.5 and higher	x	-	x	x
OSS ASN.1[19] tools	most used tool, license available for academic use	x	x	x	-
A2j: ASN to Java Stub Compiler [20]	A Pure Java ASN.1 to Java Stub precompiler (With Ant/Maven plugins) and BER encoding runtime. (no PER because from 2001, open source)	x	-	-	x
ASN1C from obj-sys [21]	closed source	x	x	x	-
GNU Libtasn1 [22]	open source, only DER	-	x	-	x
OSDT Oracle Security Developer Tools [120]	Oracle Crypto Software Development Kit offers reading and writing BER-encoded and DER-encoded ASN.1 structures.	x	-	-	x

Table 5: Available ASN.1 Tools Overview

Table 5 presents an overview of some tools available for ASN.1 encoding.

3.8.6 FIPA ACL Bit-Efficient Encoding

FIPA ACL Bit-Efficient Encoding defines bit-efficient encoding of FIPA ACL messages. It allows efficient encoding of the different FIPA ACL message parameters. One disadvantage here is that the content parameter of the message is treated as string and cannot be encoded using this standard even so it would be possible if content is structured. It is possible to extend the standard by missing data types and to apply it also to the content parameter. The FIPA standards are mainly used in the software agent research community. The most known platform here is JADE (see section 3.5.7).

FIPA ACL Bit Efficient standard (FIPA ACL BE) [8] only covers the encoding of the different message parameters and is not defined to also encode the content of the message in similar way. This means we would have to extend the FIPA ACL BE standard and apply the techniques introduced there to the content itself.

The FIPA ACL BE standard for encoding messages does not cover static encoding tables which we would need here to encode the language keywords, ontology codes, etc. The standard describes use of dynamic code tables, which have to

[19] OSS Nokalva, Inc. company web page: http://www.oss.com.
[20] A2j download web page: http://sourceforge.net/projects/a2j/.
[21] obj-sys web page: http://www.obj-sys.com/asn1-compiler.shtml.
[22] GNU Libtasn1 web page: http://www.gnu.org/software/libtasn1.

be used unidirectional, meaning for communication from A to B and from B to A two different encoding tables have to be used. The encoding of A3ME keywords, ontology, etc. could be done similar to the encoding tables covered by the standard.

After identifying ASN.1 to be more appropriate solutions for message encoding for A3ME we discontinued our FIPA ACL BE based development.

3.8.7 YAML

YAML[23] is a human-readable data serialization format used by a variety of programming languages. It is based on the assumption that most data can be represented as an array and/or a scalar. Listing 8 shows an example of variable mappings (Source [49]):

```
hr: 65 # Home runs
avg: 0.278 # Batting average
rbi: 147 # Runs Batted In
```

Listing 8: YAML example

In Java programs YAML can be used very easily via Snakeyaml [24]. It allows creating YAML description from Java basic data types and vice-versa.

In A3ME we decided not to use YAML because of its not optimal message size and the lack of validation mechanism for data defined in YAML.

3.9 Content Description Languages

There exists various technologies to describe and represent a content. Here we will shortly describe some of those which had influence on our work.

3.9.1 SensorML

The Sensor Model Language (SensorML) is an approved Open Geospatial Consortium[25] standard. It specifies standard models and an XML encoding to describe sensors and measurement processes. [51]

SensorML supports the following functions:

- descriptions of sensors and sensor systems,
- discovery of sensor and process information,
- processing and analysis of the sensor observations,
- geolocation of observed values,
- sensor performance characteristics.

SensorML is used by the Sensor Observation Service (SOS) web service to describe the sensors and the measurement processes.

SensorML results in very long descriptions since it is described in XML. Therefore even for simple sensor descriptions the resulting messages are to big for the resource-constrained devices like battery-powered sensors. Additionally the participants of the information must be capable to compose and parse xml. This would also use to much computing power and program space, which are very limited on the resource-constrained battery-powered sensor nodes.

3.9.2 IEEE 1451

IEEE 1451 [2] is a set of Smart transducer interface standards (Table 6) developed by the IEEE Instrumentation and Measurement Society's Sensor Technology Technical Committee. These standards are used in industry to describe interfaces of sensors, actuators and devices connected via a wire or wireless. An important element of these standards is a data sheet TEDS (Transducer Electronic Data Sheet) for each device. The TEDS contain identification, calibration, correction data, and manufacturer-related information.

This technology is a specialized solution for the manufacturing area. The idea that each device brings its own description and exchanges it independent of the underlying physical communication media corresponds to the A3ME goals.

[23] YAML web page: http://www.yaml.org/

[24] Snakeyaml web page: http://code.google.com/p/snakeyaml/

[25] The Open Geospatial Consortium (OGC) is an international industry consortium of 451 companies, government agencies and universities participating in a consensus process to develop publicly available interface standards. Web page: http://www.opengeospatial.org/.

1451.0-2007	IEEE Standard for a Smart Transducer Interface for Sensors and Actuators – Common Functions, Communication Protocols, and Transducer Electronic Data Sheet (TEDS) Formats [17]
1451.1-1999	IEEE Standard for a Smart Transducer Interface for Sensors and Actuators – Network Capable Application Processor Information Model
1451.2-1997	IEEE Standard for a Smart Transducer Interface for Sensors and Actuators – Transducer to Microprocessor Communication Protocols & TEDS Formats
1451.3-2003	IEEE Standard for a Smart Transducer Interface for Sensors and Actuators – Digital Communication & TEDS Formats for Distributed Multidrop Systems
1451.4-2004	IEEE Standard for a Smart Transducer Interface for Sensors and Actuators – Mixed-Mode Communication Protocols & TEDS Formats
1451.5-2007	IEEE Standard for a Smart Transducer Interface for Sensors and Actuators – Wireless Communication Protocols & Transducer Electronic Data Sheet (TEDS) Formats
1451.7-2010	IEEE Draft Standard for a Smart Transducer Interface for Sensors and Actuators – Transducers to Radio Frequency Identification (RFID) Systems Communication Protocols and Transducer Electronic Data Sheet Formats

Table 6: The 1451 Family of Standards

3.9.3 RDF

The Resource Description Framework (RDF) is a framework for representing information in the Web. The underlying structure of any expression in RDF is a collection of triples, each consisting of a subject, a predicate and an object. A set of such triples is called an RDF graph. This can be illustrated by a node and directed-arc diagram, in which each triple is represented as a node-arc-node link. A node may be a URI reference, a literal, or blank. Properties are URI references. [13]

The RDF 1.0 specification supported only XML as serialization format. The new version 1.1 of the RDF specifications [41] added further serialization formats: JSON-LG, RDFa, etc.

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:contact="http://www.w3.org/2000/10/swap/pim/contact#">
  <contact:Person rdf:about="http://www.w3.org/People/EM/contact#me">
    <contact:fullName>Arthur Herzog</contact:fullName>
    <contact:mailbox rdf:resource="mailto:aherzog[at]dvs.tu-darmstadt.de"/>
    <contact:personalTitle>Dipl.-Inf.</contact:personalTitle>
  </contact:Person>
</rdf:RDF>
```

Listing 9: RDF description of a person

RDF allows to describe semantical information and relationships between entities. In our case we can describe the classification in RDF as a set of *is-a* triples and the capabilities of a device with a predicate *has-capability*.

In A3ME we decided to use a simple predefined classification for description of devices and their capabilities instead of a full fledged semantical description like RDF to be able to use it on resource-constrained devices directly.

3.9.4 HTML Microdata

HTML Microdata[26] is a W3C Working Draft specification. This specification allows machine-readable data to be embedded in HTML documents in an easy-to-write manner, with an unambiguous parsing model. It is compatible with numerous other data formats including RDF and JSON. [93]

The specification adds the following attributes as global attributes to HTML elements:

- itemid: must have a value that is a valid URL potentially surrounded by spaces.
- itemprop: defines a property to an item inside an itemscope.
- itemref: set of IDs of elements in the same home subtree.
- itemscope: creates a new item, a group of name-value pairs.
- itemtype: specifies one or more absolute URL(s), all of which are defined to use the same vocabulary.

The following example (listing 10) shows an example describing an address.

[26] HTML Microdata web page: http://www.w3.org/TR/microdata/.

```
<div itemscope itemtype="http://schema.org/Organization">
  <span itemprop="name">Technische Universitaet Darmstadt</span><br/>
  Address: <br/>
  <div itemprop="address" itemscope itemtype="http://schema.org/PostalAddress">
    <span itemprop="streetAddress">Hochschulstrasse 10</span>, <br/>
    <span itemprop="addressLocality">Darmstadt</span>, <br/>
    <span itemprop="addressRegion">Hesse</span>.<br/>
  </div>
  Telephone: <span itemprop="telephone">06151/16-01</span>.<br/>
  <a href="http://www.tu-darmstadt.de/" itemprop="url">http://www.tu-darmstadt.de</a>.
</div>
```

Listing 10: HTML Microdata example

HTML Microdata is specialized on extension of HTML documents with semantical information and is not suitable for our purpose because of the description size and computing requirements for parsing it.

3.9.5 Microformats

Microformats[27] are small patterns of HTML to represent commonly published things like people, events, blog posts, reviews and tags in web pages. They are designed for humans first and machines second, microformats are a set of simple, open data formats built upon existing and widely adopted standards and enable simple embedding of semantics in HTML to enable decentralized development. [119]

Listing 11 shows an example for the description of an address using microformats.

```
<div class="vcard">
  <span class="fn org">Technische Universitaet Darmstadt</span>
  Address:
  <div class="adr">
    <span class="street-address">Hochschulstrasse 10</span>
    in <span class="locality">Darmstadt</span>,
    <span class="region">Hesse</span>.
  </div>
  Telephone: <span class="tel">06151/16-01</span>
  <a href="http://www.tu-darmstadt.de/" class="url">http://www.tu-darmstadt.de</a>
</div>
```

Listing 11: Microformats example

The microformats are not standardized by any standards body and are maintained by the microformats community through an open wiki. The provided microformats specifications are derived and aligned to existing established standards, e.g. person and institutions description is derived from representation of vCard (RFC2426) [73].

Microformats is specialized on extension of HTML documents with semantical information and is not suitable for our purpose because of the description size and computing requirements for parsing it.

3.10 Ontologies

Ontologies are used to formally describe classifications and relations of entities and their properties.

3.10.1 Context Related Ontologies

Preuveneers introduces in [135] an extensible context ontology for ambient intelligence, which was used for the CoDAMoS[28] Project (Context-Driven Adaptation of Mobile Services). This ontology is expressed in OWL and consists of four context areas: user, environment, platform and service. Context Broker Architecture (CoBrA) [57] is an architecture, which allows to define, publish and interpret context information under considerations of privacy policies of the involved entities. The Service-oriented Context-Aware Middleware (SOCAM) [80] is another architecture for development of context-aware mobile services with reduced resource requirements. The Multi-Sensor Oriented Context Model (SOCOM) [148] is based on ontologies and focuses on the relations between the sensors and the context. The EASY

[27] Microformats web page: http://microformats.org.

[28] Project CoDAMoS web page: https://distrinet.cs.kuleuven.be/projects/CoDAMoS/.

framework described in section 3.7 uses ontologies defined in [135] and [113] to describe the capabilities, context and QoS properties and their interconnections. In [133] service design methodology based on OWL-s is presented which allows to describe different service related properties.

The Context Ontology Language (CoOL) [147] is derived from the context modeling approach based on ontologies as facts. CoOL represents its knowledge model in OWL for easier handling and in F-Logic for querying and rule based extensibility.

3.10.2 Sensor Ontologies

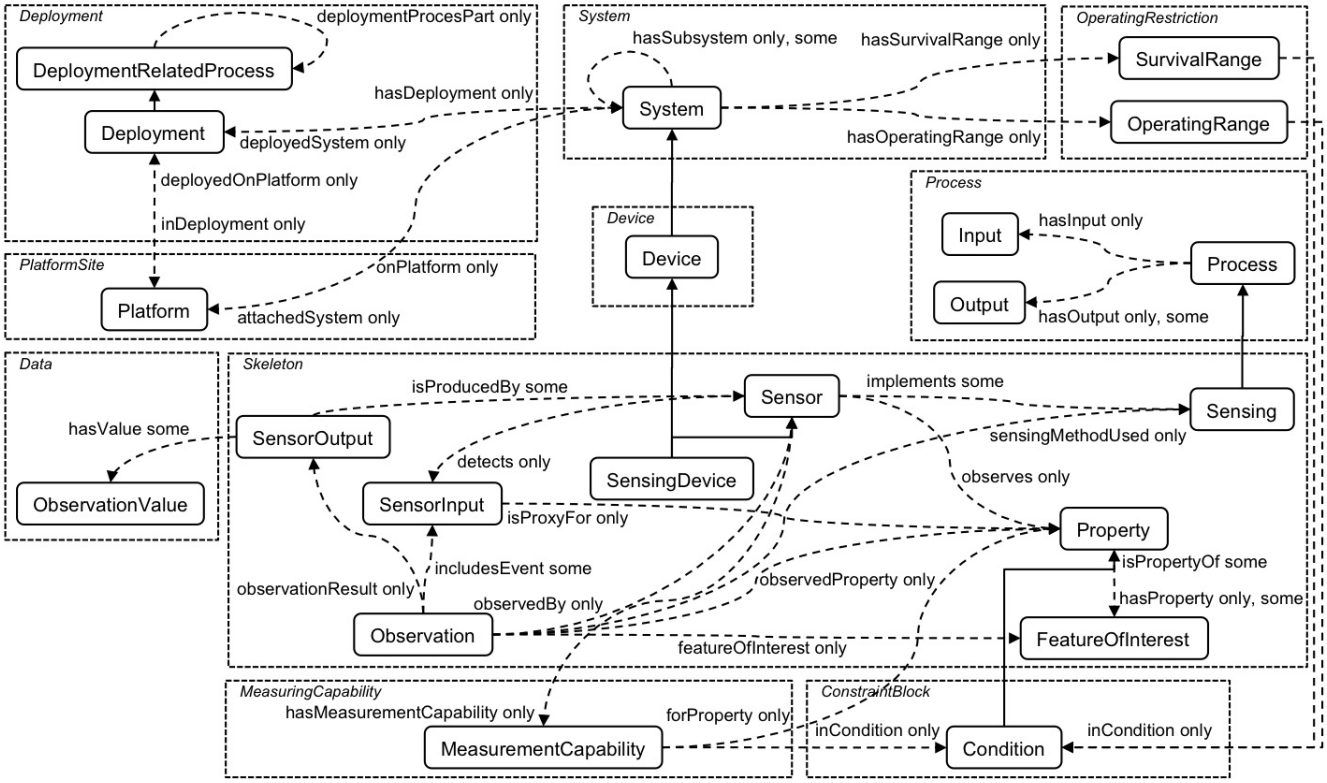


Figure 12: Overview of the Semantic Sensor Network Ontology Classes and Properties [108]

Open Geospatial Consortium (OGC®) published a series of specifications in the focus area of Sensor Web Enablement (SWE)[29] [145] [137] to define interfaces and data definitions for the integration of sensors. Each of these specifications either defines or uses ontologies for the specific task:

- Observations & Measurements Schema (O&M) [61],
- Sensor Model Language (SensorML, section 3.9.1) [51],
- Sensor Observations Service (SOS) [52],
- Sensor Planning Service (SPS) [29],
- SWE Common Data Model [30],
- SWE Services Common [31].

W3C Semantic Sensor Network Incubator Group (SSNXG)[30] evaluated various existing ontologies related to sensor networks and developed an ontology for sensor networks in consensus with the related work. We contributed to the work of SSNXG and the results of this work were published in [108]. Figure 12 shows the overview of the Semantic Sensor Network (SSN) ontology. The developed ontology is divided into several modules to group the various concepts related to sensors and allows to focus on specific aspect for different tasks. In section 4.7.5 we present an ontology based on the SSN ontology which describes the devices used for the prototypical A3ME deployment used for evaluation.

3.10.3 Other Ontologies

In [99] a Policy Language for a Pervasive Computing Environment (Rei) allows to define different kind of properties for the devices and their users.

[29] Sensor Web Enablement (SWE) working group web page: http://www.opengeospatial.org/projects/groups/sensorwebdwg.

[30] W3C Semantic Sensor Network Incubator Group (SSNXG) web page: http://www.w3.org/2005/Incubator/ssn/

A WAP User Agent Profile [155] defines the information which can be send to the server from the requesting mobile WAP device to inform the server about the device's capabilities. The information is described using RDF and can contain information about the hardware, software, networking capabilities, etc.

3.11 Content Query Languages

There exists a variety of content query languages. Here we will shortly describe some of those which had influence on our work.

3.11.1 SPARQL Protocol and RDF Query Language

SPARQL (SPARQL Protocol and RDF Query Language) is a set of specifications to access and manipulate semantic information. The data sources can be native RDF (section 3.9.3) or other information stores transformed to RDF via middleware. The syntax of SPARQL is very similar to SQL making it very easy to use. At the same time it allows to formulate very complex queries. A result of a query can be a result set or a RDF graph. [35] [36]

The simple SPARQL query in Listing 12 returns all devices and counts of their capabilities contained in the data store:

```
PREFIX a3me: <http://example.com/a3me/1.0/>
SELECT ?name (COUNT(?capability) AS ?count_capabilities)
WHERE {
    ?device a3me:name ?name .
    ?device a3me:hascapability ?capability .
} GROUP BY ?device ?name
```

Listing 12: SPARQL Query Example

Variables are indicated by a '?' or '$' prefix. The SPARQL query processor binds the variables in the query to the underlying data source and finds binding combinations which satisfy the facts from the *WHERE* part in the query.

In A3ME devices with enough resources could additionally offer a SPARQL interface to allow queries on the information available through A3ME to devices which can not use A3ME directly.

3.11.2 KQML

The Knowledge Query and Manipulation Language (KQML) is a language and protocol to query and exchange information between information sources. Information sources could be information stores or software agents. KQML was also often used as an Agent communication language. KQML's "performatives" are operations that agents perform on each other's knowledge and goal stores. [74]

In 1997 there was a proposal for a new KQML Specification [105]. But it did not became accepted as a common standard and was then superseded by FIPA-ACL and SPARQL.

A KQML message is also called a performative. Parameters in performatives are indexed by keywords and therefore order independent. These keywords, called parameter names, must begin with a colon (:) and must precede the corresponding parameter value. Performative parameters are identified by keywords rather than by their position because there are many optional parameters to performatives. Listing 13 shows the common structure and an example of a KQML message.

```
(<Performative>
    :content    <speechact>
    :sender     <name>
    :receiver   <name>
    :language   <text>
    :ontology   <text>
)
```

Listing 13: Common Structure of a KQML Message

And the Listing 14 shows a corresponding example message [105].

```
(tell
   :sender       B
   :receiver     A
   :in-reply-to  id1
   :reply-with   id2
   :language     Prolog
   :ontology     foo
   :content      "[bar(a,b),bar(c,d)]"
)
```

Listing 14: Example of a KQML Message

KQML was added here for historical reasons. It has been made obsolete by FIPA-ACL and SPARQL, and is therefore not used in A3ME.

3.11.3 FIPA-ACL

Agent Communication Language (ACL) is a standard language for agent communications proposed by the Foundation for Intelligent Physical Agents (FIPA) (see section 3.5.6). It is composed of three standards:

- FIPA Communicative Act Library Specification [10],
- FIPA ACL Message Structure Specification [9],
- FIPA Interaction Protocol Library Specification [4].

Parameter	Category of Parameters	Description
performative	Type of communicative acts	Denotes the type of the communicative act of the ACL message
sender	Participant in communication	Denotes the identity of the sender of the message, that is, the name of the agent of the communicative act.
receiver	Participant in communication	Denotes the identity of the intended recipients of the message.
reply-to	Participant in communication	This parameter indicates that subsequent messages in this conversation thread are to be directed to the agent named in the reply-to parameter, instead of to the agent named in the sender parameter.
content	Content of message	Denotes the content of the message; equivalently denotes the object of the action. The meaning of the content of any ACL message is intended to be interpreted by the receiver of the message. This is particularly relevant for instance when referring to referential expressions, whose interpretation might be different for the sender and the receiver.
language	Description of Content	Denotes the language in which the content parameter is expressed.
encoding	Description of Content	Denotes the specific encoding of the content language expression.
ontology	Description of Content	Denotes the ontology(s) used to give a meaning to the symbols in the content expression.
protocol	Control of conversation	Denotes the interaction protocol that the sending agent is employing with this ACL message.
conversation-id	Control of conversation	Introduces an expression (a conversation identifier) which is used to identify the ongoing sequence of communicative acts that together form a conversation.
reply-with	Control of conversation	Introduces an expression that will be used by the responding agent to identify this message.
in-reply-to	Control of conversation	Denotes an expression that references an earlier action to which this message is a reply.
reply-by	Control of conversation	Denotes a time and/or date expression which indicates the latest time by which the sending agent would like to receive a reply.

Table 7: FIPA ACL Message Parameters

FIPA Communicative Act Library Specification defines 22 types of messages – performatives (Table 2). The idea behind it is that each message has some intention and each performative represents one class of intentions. Therefore each message is tagged with a performative which defines its purpose e.g. request.

The FIPA ACL Message Structure Specification [9] defines the different parameters a message can have (Table 7). The only required parameter is the performative, which assigns one of the 22 performatives to the message.

In our work we adopt the idea to use performatives for different message types. Furthermore we added the message parameters defined in the FIPA ACL Message Structure Specification [9] as optional parameters to the A3ME messages.

3.11.4 Simple Sensor Interface

Simple Sensor Interface (SSI) protocol is an application layer protocol for exchange of data between sensors and terminals. SSI was developed under the lead of Nokia up to 2006 and was used within the MIMOSA project (section 3.2.5). The protocol is designed to be simple and to have small footprint, and can be utilized with UART, TCP/IP or nanoIP. The protocol defines 18 different commands which can be exchanged between the sensor and the terminal. [96] [107, p. 15]

The commands defined in the SSI are very specific for use with sensors. In our framework we use A3ME Query Language encoded in ASN.1. allowing to define messages for interactions not just with sensors but for broader range of interactions between devices.

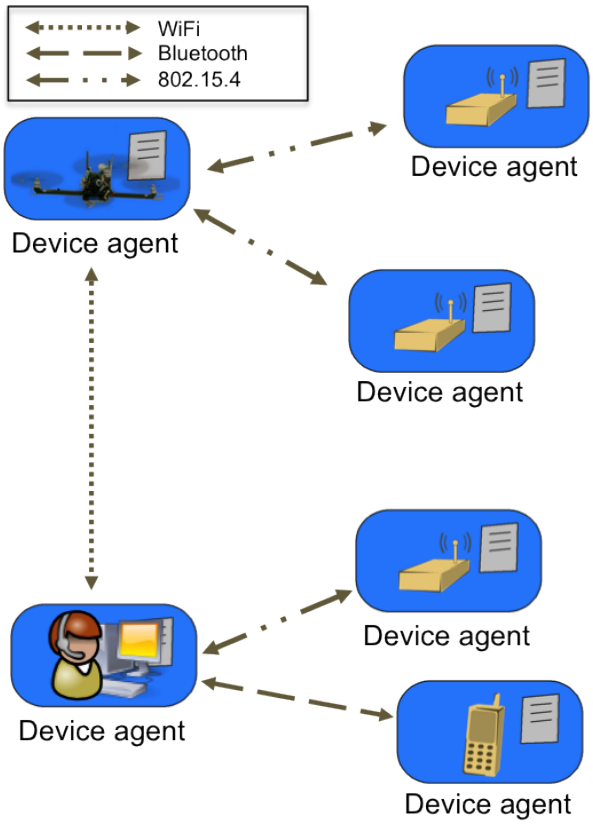


Figure 13: A3ME Device-Agents

In this chapter we describe the A3ME framework developed in this thesis. The name A3ME is an acronym for ”Device-Agent based Middleware for Mixed Mode Environments“ where ”MMM“ is abbreviated to ”3M“. Mixed Mode Environments (MME, section 1) are environments with different dimensions of heterogeneity and present challenges described in section 1.1 to a middleware to operate there. The A3ME framework is a solution to deal with these challenges. Figure 13 shows an example of a MME with four types of devices and three different communication technologies. An early draft of our solution was described in [92].

First we present the overall A3ME System Architecture (section 4.1). In section 4.2 we describe the representation concept for the individual devices – device-agents. Then in section 4.3 we introduce Device-Agent Interfaces (DAI), which enable device-agent interactions by exchange of messages. To realize DAI device-agents must be able to communicate with each other (section 4.5). Once the physical communication is possible, we discover whether the other devices are A3ME capable by sending them a self-introduction and a device discovery message. For this we use the A3ME Query Language (`A3ME-QL`, see section 4.11). The DAI (see section 4.3) defines the basic mechanisms to enable agents to publish, discover and use `services`. For describing these services, the data and the tasks we use a `content representation` schema described in section 4.10.

4.1 A3ME System Architecture

To enable interactions between heterogeneous devices as described in section 1.1 we follow basic principles:

1. Neutral data representation,
2. Technology independent messages and
3. Technology independent message exchange.

Figure 14 shows the A3ME system architecture.

4.1.1 Neutral Data Representation

To enable neutral data representation we developed a common neutral semantical base to describe the individual parties and their capabilities and properties. Furthermore we added the neutral representation for addressing and for the different types of data.

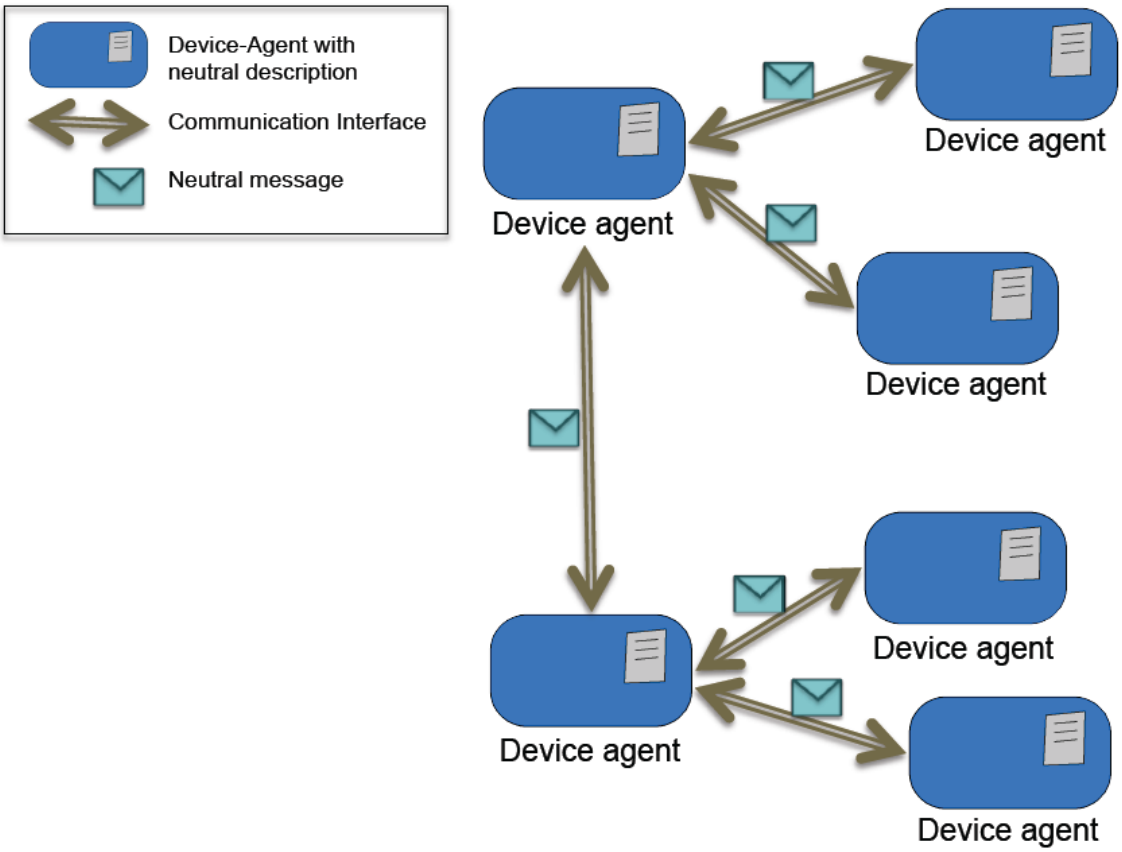


Figure 14: A3ME System Architecture

The use of neutral data representation enables our framework to convert the descriptions of devices, their capabilities, properties, services and data into a neutral form and back from and to the individual technologies used. This reduces the amount of required transformation of data representations between the different technologies from $O(n^2)$ to $O(n)$. Without a neutral representation the amount of transformations (T) between n technologies can be calculated using the formula $T(n) = n(n-1)/2 = O(n^2)$. When using a neutral representation, the amount of transformations is $T(n) = n = O(n)$. Figure 15 demonstrates the amount of transformations between four different technologies.

In A3ME each device is represented by a device-agent which offers the neutral description and interaction possibilities of the device to others. The description is based on the classification described in section 4.7.

4.1.2 Technology Independent Messages

To be able to exchange messages via different communication technologies we define a generic message structure (section 4.8). Here each message is assigned with a Message Performative, which expresses the purpose category of the message, allowing basic handling of the message based on its performative.

The content of the message is defined in section 4.10. For the definition of the different messages we defined an ASN.1 message definition schema, which is used to define concrete messages. The different information entities used here are described using the A3ME Classification. For the transport of the messages we use ASN.1 unaligned packed encoding rules (PER) in section 4.13.

4.1.3 Technology Independent Message Exchange

The second basic principal in the A3ME framework is the exchange of messages independent of the communication technology. This means we can use any available communication technology available on a concrete device to exchange messages. For this the messages are transported as payload in the different communication technologies like LAN, WLAN, Bluetooth, etc. For more details see section 4.5.

The different communication technologies might introduce specific limitation on reliability, bandwidth, delay etc. These specifics are seen as characteristics of the individual communication interface and can be described accordingly.

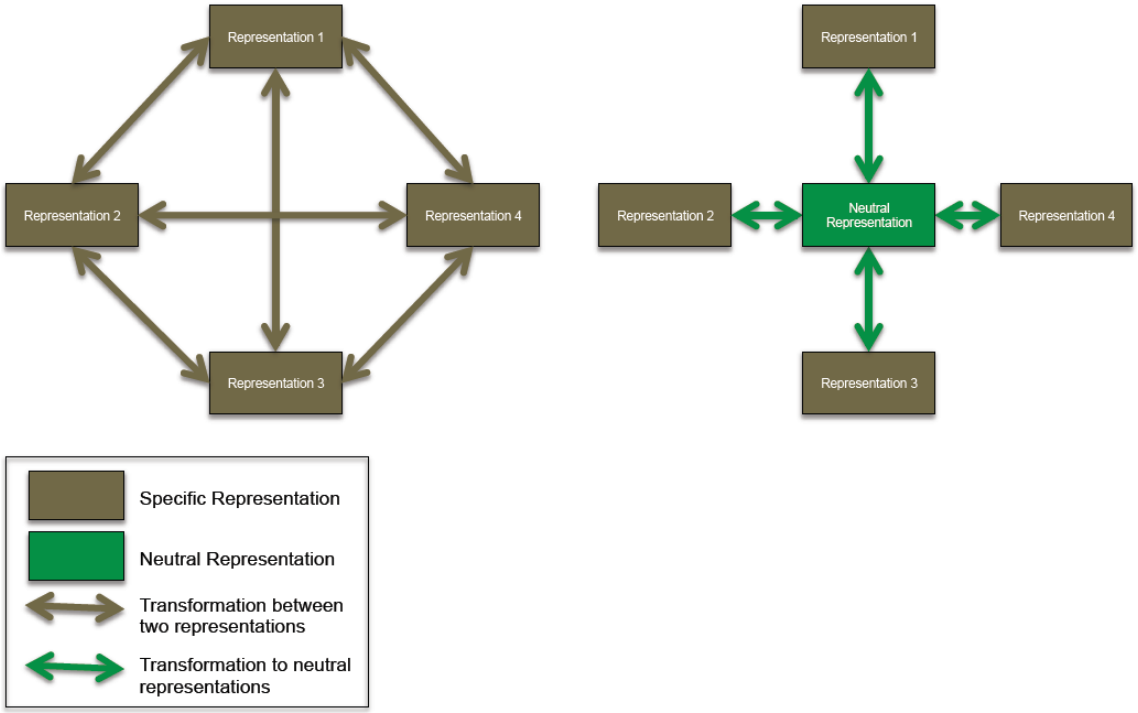


Figure 15: Amount of Transformations Required for 4 Different Technologies without and with a Neutral Representation

4.2 Device Representation

To represent the heterogeneous devices in and specially across different domain environments we use the notion of an agent. Each device in the network is represented as a device-agent. A device-agent knows the capabilities, properties, constrains and policies of the device-entity it represents. The inner representation and realization of the device-agent is not visible to the outside. This allows our middleware to represent any kind of device with any properties, including also devices and properties which will be introduced in the future. The agent abstraction for the various devices considers them just as different entities with individual properties and capabilities which can offer and use services.

To distinguish our agents from the different kinds of agents used in computer science, we call the agents in our approach `device-agents`. A device-agent is a special software, which resides on a specific device and knows the device's specific capabilities, constraints, policies and services (Figure 13). Each of the nodes represented as device-agent is an independent entity, which is also able to function on its own. These entities interact with each other to build a network and to enable higher level services and capabilities. Agent-based approach facilitates the self organization and adaptability of the system.

Our understanding of an agent differs significantly of that used in current WSN projects like Agilla, described in [76]. In Agilla an agent represents code which can move between devices and continue its execution. In our work, agent means an abstraction of a network node.

Each device-agent is seen as black box, which means we want to hide the hardware and software the node is built of, and just show its capabilities and services to the outside (see Figure 16). To interact with each other, device-agents have to support a basic set of messages and interaction protocols described in more detail in the following sections. The internal software architecture of a device-agent is described in section 4.6.

4.3 Device-Agent Interface

In a general case we have many different kinds of devices represented by device-agents. All these device-agents must provide a Device-agent Interaction Interface (DAI) to interact with each other directly. DAI defines a basic set of Interaction protocols and the message structure independent of the hardware and communication technology. Implementation of DAI is hardware specific for each type of device and provides all basic hardware dependent capabilities of the current device and can offer them as services.

Additionally each device can have one or multiple applications running on it. Applications interact with other nodes in the network through DAI. Applications can also define tasks for the agent and additional hardware independent services like aggregation functions. These hardware independent services are also offered through DAI (Figure 16). The services offered can be publicly accessible by other agents, restricted for use by only some agents or by authentication or they can be private, what means usable by local applications only.

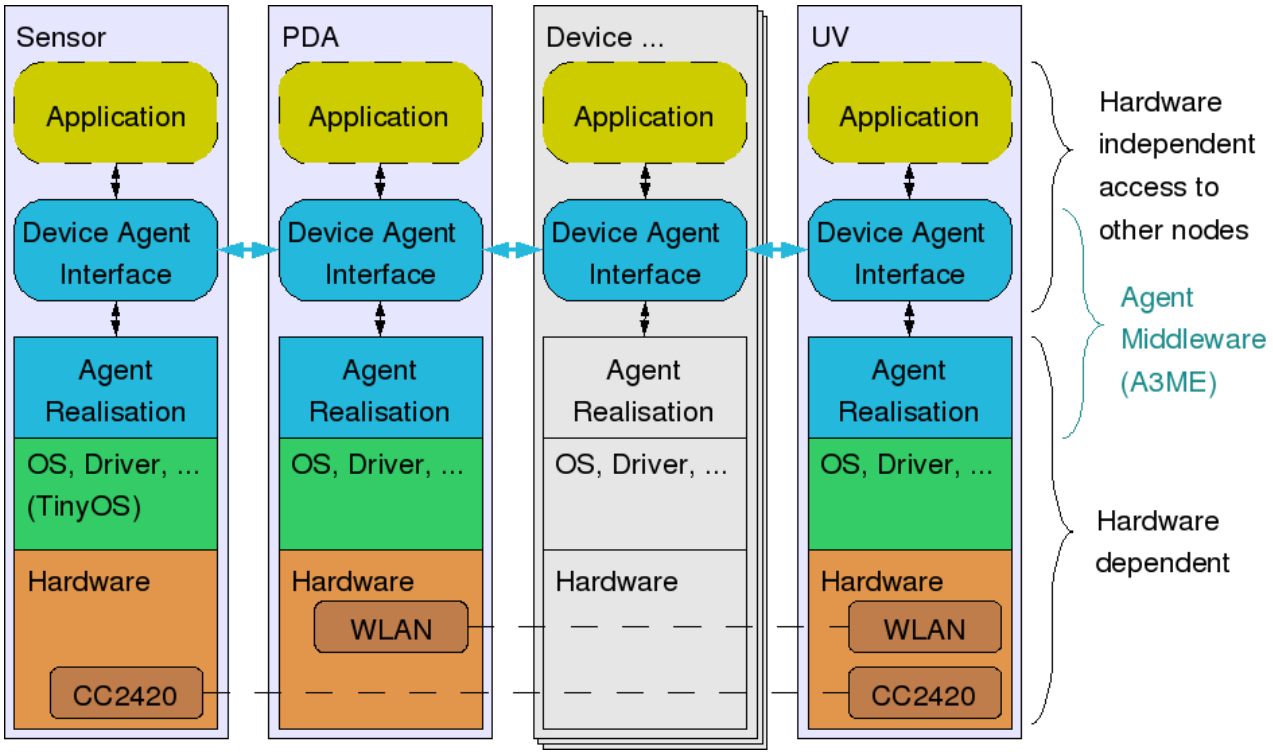


Figure 16: The Individual Device-Agents in A3ME

The device-agent interface can be realized at three levels: core, basic and extended (Figure 17). Where basic and extended levels are built on top of core DAI.

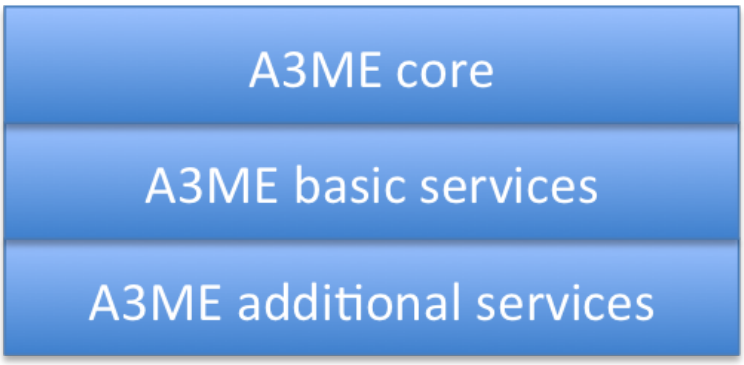


Figure 17: A3ME Device-Agent Interface Levels

The basic goal of A3ME is to enable interactions between different devices, this is what the core device-agent interface provides. The A3ME DAI core offers the basic information interactions with other device-agents: It provides the ability for self-description and the ability to exchange and handle A3ME messages. A3ME core is mandatory for an A3ME device-agent.

The basic DAI level offer services, which cover the hardware functionalities of the node, e.g. in the case of a sensor node all its sensor readings are offered as services and for an actuator the basic actuation commands.

The extended DAI level offers additional services allowing applications of a device to register other kind of services (e.g. web services) and make them searchable through A3ME.

All interactions happen through exchange of messages. The structure of these messages is described in 4.8. Different interactions are composed of interaction primitives described in 4.9. In section 4.10 we describe how the content of the A3ME messages is built.

How the messages are really exchanged between the devices depends on the available communication components of the device. The A3ME messages are transmitted as payload using any communication technique. How the message was transmitted or has to be transmitted is contextual information to the message.

4.4 Device Description

Important part of the A3ME framework is the capability of each device to describe itself and to exchange the device description with other device-agents. To allow to describe the different device types, capabilities, services, properties and data we developed a generic neutral classification described in section 4.7. These allows to describe each device and its capabilities and properties by referencing the A3ME classification. Table 8 shows an example description of the TelosB sensor node using references to the A3ME classification.

Described item	**Name**	**Referenced A3ME Classification**
Device type	TelosB sensor node	a3me.device.mote
Processor	MSP 430 microcontroller	a3me.capability.cpu
Storage	RAM memory	a3me.capability.storage.ram
Storage	Flash memory	a3me.capability.storage.flash
Communication	IEEE 802.15.4 compliant CC2420 radio	a3me.capability.communication
Communication	USB interface	a3me.capability.communication
Sensor	temperature	a3me.capability.sensor.temperature
Sensor	visible light	a3me.capability.sensor.light
Sensor	visible and infrared light	a3me.capability.sensor.light
Sensor	humidity	a3me.capability.sensor.humidity
Sensor	battery voltage	a3me.capability.sensor.voltage
Sensor	user defined switch	a3me.capability.sensor.switch
Sensor	battery voltage	a3me.capability.sensor.voltage
Power supply	2 AA batteries	a3me.capability.energy
Power supply	via USB	a3me.capability.energy
HID	red LED	a3me.hid.output
HID	blue LED	a3me.hid.output
HID	green LED	a3me.hid.output

Table 8: Description of the TelosB Sensor Device using References to the A3ME Classification.

Use of a neutral description in combination with communication technology independent representation (4.10) allows to reduce the amount of required transformations between different technologies/frameworks from $\frac{n(n-1)}{2}$, when every technology has a transformation to every other technology, to n transformations to a neutral representation.

In A3ME most interactions are based on the neutral classification, e.g. request of temperature sensor data. These allows to request information from a priori unknown devices.

4.5 Communication

Before any message-oriented device interactions can take place the devices must be able to communicate with each other. In a heterogeneous environment, where a variety of different communication technologies is used, it is not feasible to focus or prefer a specific one. The communication in general should be seen as the process to exchange messages: to send and to receive messages. Therefore a middleware for a heterogeneous environment should use generic communication interfaces, which then can be realized by the different communication technologies.

Each device must have at least one communication capability[31]. The different communication capabilities are represented inside the device-agent as separate communication interfaces, with their individual properties. Directly reachable communication partner can be used to forward the messages: not only in the same communication technology but also to other communication interfaces.

4.5.1 Device Discovery

In general before communication[32] can take place, a device needs to discover it's neighbors. The discovery of the neighbors is closely related to the communication technology used. Many communication technologies already have device discovery protocols on the physical layer or data link layer, which can be used here. In the cases where the communication range is limited (e.g. LAN, most wireless communications) the discovery is done by broadcasting a hello message and waiting for responses from other devices which heard the hello message (e.g. mDNS [58]). If the communication range is not directly limited the broadcasts are usually limited, to avoid infinite propagation and pollution to the communication medium.

[31] Devices with no communication capability are not considered here.

[32] An exception might be an unidirectional communication e.g. in the case of a beacon, where the discovery of communication partners might not be required.

4.5.1.1 Neighbor Discovery

The various neighbor discovery techniques described in section 3.6 can be characterized using following dimensions:

- Communication network,
- Communication channel,
- Radio sleep/awake behavior and
- Communication protocol.

For devices with multiple communication interfaces the device discovery is done on each communication network independently, since most neighbor discovery techniques are designed for a single specific communication network (section 3.6). In a communication networks where the communication channel is not fixed, is periodically changed or can dynamically change over time, additional mechanisms are introduced to ensure the discovery of the neighbors on all channels. In resource constrained networks like WSNs the individual nodes might be asleep most of the time to save energy. This means that their communication interface is switched off and they can not receive any messages in these sleeping phases, so it must be ensured that the sending and receiving nodes have a mechanism to meet in a common time interval to communicate. Another dimension is the protocol used on top of the data link layer. Here the devices often can not discover each other even so they use compatible communication hardware. Here it is theoretically possible to have a device which supports multiple protocols and switches to the right protocol to communicate with the individual neighbors.

By default the native mechanism(s) of the used communication technology should be used for device discovery. For communication technologies, where the number of directly reachable communication partners is not limited by spacial location in general no device discovery should be triggered automatically. The triggering, specification and filtering of the communication partners here should be controlled by higher level application which use the framework.

The device discovery can be started on a request from the user, from a higher level application or from another device. Otherwise each device only passively collects the information about its neighbor from the received messages. A device advertises itself when it is switched on and can also do it periodically, e.g. to inform it's neighbors in a dynamic network about its presence.

In our prototypical implementation (section 5) we use three different communication techniques:

- [Com-1] Bluetooth version 3.0 between the smartphone and the workstation.
- [Com-2] IEEE 802.15.4 compliant radio between the TelosB sensor nodes and the base-station,
- [Com-3][33] IEEE 802.15.4 compliant radio between the Sun Spot sensor nodes and the base-station.

For Bluetooth *[Com-1]* we use the build-in Bluetooth neighbor discovery (see section 3.6.1). Bluetooth uses frequency hopping on 79 channels and the challenge is to meet on a common channel to discover each other. The discovery is either triggered:

- by the user,
- on startup of the A3ME software on the corresponding device or
- implicitly through a request, asking for the different devices.

Additionally the discovery might be triggered periodically by individual devices. For the configuration of the cycles for discovery and discoverable modes one of the algorithms described in section 3.6.2 could be used.

The communication interface [Com-2] uses a common frequency channel for all nodes in a concrete WSN making it unnecessary to discover the frequency. The used B-MAC protocol [132] allows to send and receive messages without previous neighbor discovery. To achieve this the protocol uses carrier sense media access capability and an adaptive preamble sampling scheme.

[Com-3] uses the same communication hardware and similar mac protocol as [Com-2] which allows to send at any time to all neighbors without an explicit neighbor discovery. Therefore the neighbor discovery here is done simply by sending out a request and all neighbors in range answer this request.

4.5.1.2 Reachable Communication Partners

For some communication technologies like radio-, light-, or sound-based transmissions the communication range is naturally limited by physics law. The communication range here can be calculated for given parameters of used media, frequency, signal strength, etc. While for the Near Field Communication (NFC) the communication range is just a few cm, for WiFi it is about 100 m. In the case of light-based communication it is also required to have a direct line of sight between the communication partners. This limited communication range also limits the number of devices in direct communication range.

For other communication technologies the number of direct communication partners is not limited. For the IP based communication in the internet it is usually possible to communicate with any other device which is also connected to the internet.

Dependent on the used communication technology it might be feasible to discover all available communication partner in direct communication range (e.g. for NFC, Bluetooth, WiFi, Infrared light). In the case of internet it might be more feasible to select the potential communication partners by the geographical location, the IP address or otherwise.

[33] Com-2 and Com-3 are not compatible on the protocol layer even so they use compatible communication hardware.

Besides the technical compatibility of the used communication hardware, the number of reachable communication partners might be further limited by the supported protocols, encodings or policy limitations of the individual devices. Therefore only devices capable and willing to exchange messages are considered as reachable communication partners.

Directly reachable communication partner can be used to forward the messages: not only in the same communication technology but also to other communication interfaces. This increases the number reachable devices significantly.

4.5.1.3 Potential Interaction Partners

Usually only a partition of the reachable communication partners belongs to the group of reasonable interaction partners. This set might be dictated by the current task, application, ownership, reliability, properties, position, etc. of the individual devices.

4.5.2 Device Addressing

In a heterogeneous environment it is not clear what to use as an ID for a device. A device might have various communication capabilities and for each of those it has a different address often even of different kind. These addresses can vary in their form, size, uniqueness, validity range and duration. For example a device can have an IP-address for the Ethernet connection, a second IP for the WIFI connection and a comport to communicate with a sensor node base station, which on its side has an address inside the WSN.

In JADE (Java Agent DEvelopment Framework, see section 3.5.7) build in compliance with FIPA specifications agent IDs (AID) for identification of agents are used. The AID is composed of a textual agent name and the agent-platform name. Inside the AID different network addresses can be stored correspondingly to the communication capabilities of the agent.

In A3ME we decided to use a similar approach. With the difference that the ID of a device-agent (DAID) is composed of the platform name and a device-agent name instead of an agent-platform name, since in our case a device is usually not registered at anything like an agent-platform. We use an communication technology independent Device-Agent-ID (DAID) for device addressing, which contains the technology dependent addresses of the device. DAID is composed of the device type and a device-agent name.

For example the device-agent name for a workstation (ws:) named aherzog-mb would be "*ws:aherzog-mb*". Example in listing 15 shows an example DAID for a workstation. It contains the MAC-address, local IP Address, the EUI-64 address of the connected SunSpot device and the Bluetooth address.

```
daid {
  name "ws:aherzog-mb",
  addresses {
  {
    address-type "MAC",
    address "00:22:41:35:2f:db"
  },
  {
    address-type "IP",
    address "192.168.231.111"
  },
  {
    address-type "EUI-64",
    address "C0A8.E76F.0000.C22B"
  },
  {
    address-type "bt",
    address "0023123A50E0"
  }
}
```

Listing 15: Example of an Device-Agent ID (DAID) containing multiple addresses.

An DAID name is not necessarily unique. Therefore to compare two DAID it might require to compare also the contained addresses or at least one of the globally unique addresses. In the example (Listing 15) the unique mac address could be used for comparison. In cases where a device has no globally unique addresses the ucodes from the Ubiquitous ID Architecture [102] could be used. It is the same technique as used with RFID and barcodes: each entity such as a device or a location gets a unique ID and at some repository is described what the entity belonging to this ID is. In the case of Ubiquitous ID there is a central database where the information for the ucodes and the entities they belong to are

maintained. Instead of using a central repository it is also possible to add a URI to the DAID pointing to a resource with detailed description of the represented device.

4.5.3 Neutral Message Transport

Since we are not focusing on specific communication technologies the transport of the messages should happen irrespective of the concrete communication technology. In practice it means that the messages are transported as payloads in the messages of one of the available communication interfaces. The payload is generated from the A3ME messages by encoding them with ASN.1 PER (Section 4.13) to a byte array. For some communication interfaces it might be required to fragment the payload for transmission in multiple messages and recompose it again at the receiver. The specific communication technology dependent messages usually add additional information like header and trailer to the payload. Concrete realizations are described in the section 5.

4.5.4 Self Organization

The nodes in a MME have to be organized. Since we have to deal with very different kinds of networks there can not be one single solution which fits all. It is more likely that for different groups of nodes different types of network organizations are more appropriate. Possible organizations can be server client, peer-to-peer, clusters, etc. In the case of WSNs different organization models are discussed in [94].

No matter what kind of organization is used for individual nodes, it can not be static in our scenario, because nodes can appear and disappear at any time either as a matter of disconnection, failure or just because the node is moving through the environment. This means the organization has to be done on the fly and it must be adjusted to changes in the network. Each node must find out what other nodes it is or can be connected to. In the case of a wireless communication connected nodes correspond to the nodes which are in communication range. After the nodes in communication range are identified some organization pattern can be applied. The roles of the single nodes in the organization should be made dependable on the abilities and properties of the nodes.

4.5.5 Bridging of Messages Between Different Communication Interfaces

Devices with more than one communication capability like the workstation in our evaluation scenario (section 6.2.1) can and should serve as communication bridges by forwarding the messages not only on the same communication interface (com-interface) but also to the other com-interfaces if appropriate.

The messages should be forwarded if they are not limited to a specific set of devices. If the set of devices is limited, then the messages usually should only be forwarded to the corresponding com-interfaces. For example if the incoming message is addressed only to Bluetooth devices, it only should be forwarded on the Bluetooth com-interface and not bridged to other com-interfaces. But in some cases it might be also reasonable to forward the messages to other devices, which are capable to forward the messages to the targeted devices, e.g. a smartphone can bridge the requests of another smartphone addressed to sensor devices to a workstation which can communicate with those sensor devices.

The forwarding of messages might be limited in terms of range from the originator of the message. If this forwarding range is exceeded the messages also don't have to be bridged to other com-interfaces.

The bridging of messages can be prevented or limited by local policies in general or under specified conditions. The policies might be for example: bridge only messages from and to devices belonging to a specific company or do not bridge any messages if the battery level is below 20%.

4.5.6 Interactions with other Frameworks

When different frameworks have to be connected, the data representation and the messages have to be translated to and from the others framework format. Implementation of these translations for all possible combinations of frameworks and technologies would be very inefficient. As described in section 4.1.1 implementation of a translation to and from a neutral representation has the following advantages:

- The number of required translations is equal to the number of frameworks. In contrast without a neutral Representation the number of translation has the complexity $O(n^2)$ (Section 4.1.1).
- For each framework only the translation to and from the neutral representation to the own framework must be implemented, while
- The translations for the other framework is done by the experts of the corresponding frameworks.
- Any new framework only has to implement the translation to the neutral representation and will then be able to interact with all the other framework, which already support the neutral representation.

In these work we consider the A3ME data structure and messages as the neutral representation which can be used to interconnect different frameworks.

In section 5.3.5 we demonstrate the interaction of the A3ME framework with UPNP.

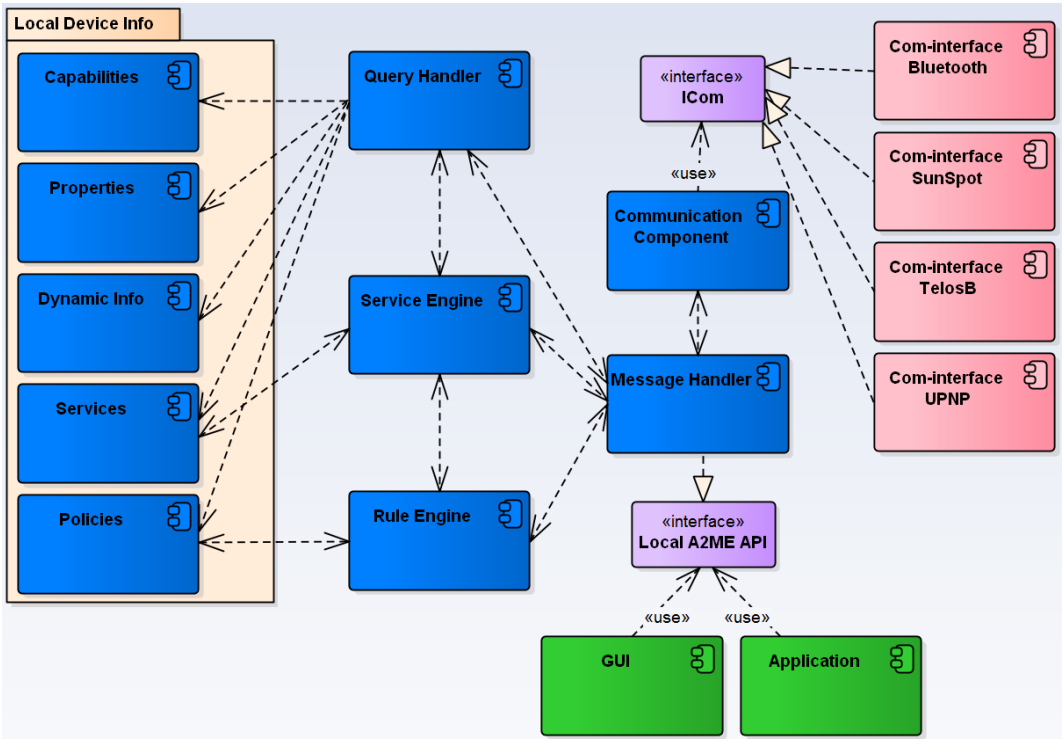


Figure 18: Architecture of an Individual A3ME Device-Agent

The abstract Device-Agent architecture is divided into following components (Figure 18):

- Communication components,
- Message handler,
- Local device info,
  - Capabilities,
  - Properties,
  - Services,
  - Policies,
  - Dynamic Info,
- Query handler,
- Service engine,
- Rule engine,
- Local A3ME API,
- GUI (optional),
- Application (optional).

4.6.1 Communication Interfaces

Each communication capability of a device is represented through a Communication Interface Component, which implements the com-interface. A Communication Interface Component covers the lower three layers from the ISO/OSI model. This means this component has to break the message down into smaller packets, if the message is to long to be transported as one packet and it is also responsible for reassembling the incoming message parts to a complete message again. The com-interface offers basic operations:

- Send function and
- MessageReceived callback function.

4.6.2 Message Handler

Message handler covers the transport layer from ISO/OSI model. It buffers and handles incoming and outgoing A3ME messages. It gets the messages from the communication interfaces and decides to which component the message has to be forwarded to handle it. In other words: it parses the incoming messages and depending on the messages' content invokes following operations:

- Updates the Local Device Info,
- Forward the query to query-handler and when getting the result send the answer,
- Composes and sends a message(s),
- Forwards the message to one or multiple Communication Components,
- Forwards the message to another component.

4.6.3 Local Device Info Handler

We distinguish different types of local info:

- Capabilities info,
- Properties info,
- Service Directory,
- Policies and
- Dynamic info.

4.6.3.1 Capabilities Info

Here the information about the capabilities of the device are stored:

- Device Type,
- HW capabilities,
  - Com-interfaces,
  - Sensors,
  - Actuators,
  - Power supply,
  - Computing power,
  - Memory,
  - Storage capacity,
  - etc.

4.6.3.2 Properties Info

Contains info about the owner, manufacturer, deployment data, etc. as key-value pairs.

4.6.3.3 Services Directory

Contains the information about available services, their availability and status. The format we let open for the individual implementation.

4.6.3.4 Policies

Contain rules for the current device. The format for the definition of the rules we let open to the individual implementation.

4.6.3.5 Dynamic Info

Holds data which is collected or received from other device-agents. Since this info is subject for aging a timestamp is added to each info item to know its age. Typical Dynamic Info stored here is:

- List of known neighbors and their known addresses
- A given number of last Conversation IDs for each sender from received messages together with timestamps of their reception.
- All known information from this device (optional, depends on the storage capabilities).

This information is used for recognizing already known messages, to avoid forwarding and handling of duplicates. The stored addresses of the device are used when answering a query from this device. Furthermore the information can be used for routing, cashing of information, etc.

4.6.4 Query Handler

Query handler gets the query request messages from the message handler and computes the result for the queries. When ready the result is forwarded to the message handler. In case of service calls the service call invocation is forwarded to the service handler.

4.6.5 Service Handler

Service handler is in charge of management and invocation of services on request or periodically (e.g. for continuous queries). The form services are realized and mechanisms for their invocation on the device we let open for the individual implementation.

4.6.6 Rule Engine

The rule engine enforces the defined policies for the device. These policies can be static for given device or can be defined dependent on various conditions. Here are examples of rules defined for a device:

- ECA (Event-Condition-Action) rules – e.g. on battery status change, if power status is less 10%, stop all bridging services.
- EA (Event-Action) rules – e.g. update battery status every 5 minutes.

The format of the rule definition and mechanisms for the execution of the rules we let open for the individual implementation.

4.6.7 Local A3ME API

Local A3ME Application Programming Interface (API) provides an interface for local applications to use the A3ME framework. It allows local applications to use and exchange local A3ME information and information available from other devices, and to interact with other devices through the A3ME framework.

4.6.8 GUI

GUI (Graphical User Interface) is a special application connected through A3ME API, which can be implemented on a device-agent if applicable and allows the user to interact with the A3ME framework. In section 5.3.2 we describe the GUI for a workstation.

4.7 A3ME Classification

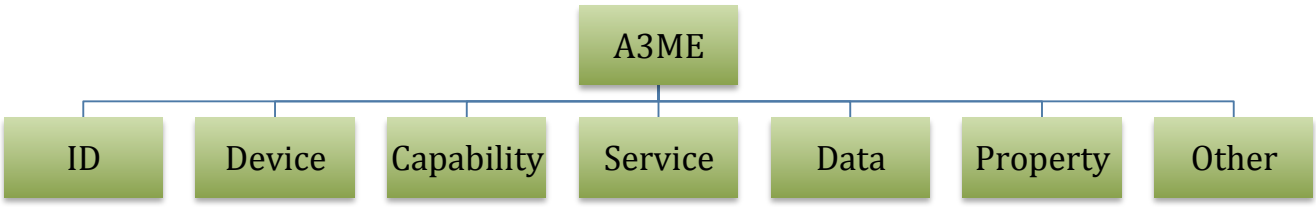


Figure 19: First Level A3ME Classification

During the investigation for this work we first looked for some existing ontology or classification, which would enable us to describe the devices and their capabilities. All identified existing classifications at that time did not match the requirements, so we had to develop a new classification to describe devices, capabilities, properties, etc. for A3ME.

First we explain the relationship and the differences between an ontology and a classification. A classification is usually used to describe is-a relationships between terms or entities hierarchically starting at the root with a generic term and getting more precise with each level down in the hierarchy. An ontology is more powerful. It is used to describe semantical relationships between terms/entities. The relationships described here can be very different and can be themselves described in ontologies. A classification therefore is a special ontology which uses is-a relationships and is hierarchically organized.

Classification and ontologies in general can be described using different content description languages:

- XML (section 3.8.1),
- RDF (section 3.9.3) a specialized XML-based description format used to describe triples of subject, predicate and object,
- OWL (Web Ontology Language) [33] specialized XML/RDF-based description format for ontologies.

We use the OWL description language to describe the A3ME classification.

During the investigation in this area we identified the Semantic Sensor Network Incubator Group (SSNXG)[34] from the World Wide Web Consortium (W3C)[35], which was working on the ontology for sensor networks. I became member of SSNXG and worked together with the group on the development of a Sensor Network ontology. The work done in this group is published in [108] and [60].

For the A3ME classification [90] we decided for the top-down strategy. This way the classification is more general as if we would drive it bottom-up by specialized use cases. The different kinds of devices are classified into classes of devices with common characteristics. For example, all kinds of cellphones, PDAs and smartphones can be classified as mobile phones.

[34] W3C Semantic Sensor Network Incubator Group (SSNXG) web site: http://www.w3.org/2005/Incubator/ssn/.

[35] World Wide Web Consortium (W3C) web site: http://www.w3.org.

In the second step, the capabilities are classified into groups of related capabilities, e.g. sensing capabilities, actuator capabilities, etc. These are then further subclassified into more specific subtypes. Since MMEs are intrinsically heterogeneous and dynamically changing, it is obvious that the classification will not be complete. This means there always will be devices and capabilities that are not covered by this classification. Therefore, extensibility of the classification is essential. Never the less for any extension the basic classification still has to stay valid.

In MME it can not be counted on a reliable connection and availability of any server. Therefore in A3ME a decentralized approach is used. Consequently the classification must be available at the devices themselves, at least the subset of the classification that is relevant for the device. Furthermore the classification must be usable on resource-constrained devices like TelosB sensor motes [14].

Concerning the constrained communication in terms of bandwidth and energy usage, the overall size of the classification should be kept small. Our goal is to encode any classification item in one byte. Because of the limited computation and storage capabilities of some devices in MME, the classification complexity in terms of its depth should also be kept low.

4.7.1 Predefined Classification

First level of the classification deals with different aspects needed to be classified in MME. Those are IDs, devices, capabilities, services, data, properties and other (Figure 19). Some of these are further subclassified. Figure 21 shows the complete classification. In Appendix A.1 the static encoding of the classification is listed. The classification is also available as a Web Ontology Language (OWL) [33] file at `http://www.dvs.tu-darmstadt.de/staff/aherzog/a3me/a3me.owl` and serves as the central URI for the predefined A3ME classification and it's extensions.

4.7.1.1 IDs

In MME there can be various networks, each using its own addressing scheme to identify the participating nodes. Therefore, we added a simple classification for the IDs. ID is further subclassified into

- Local IDs and
- Global IDs.

4.7.1.2 Devices

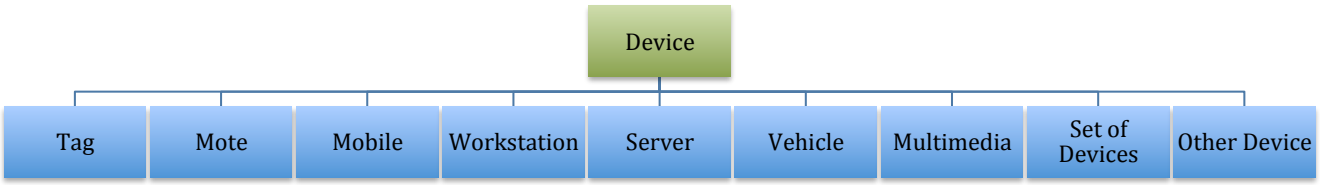


Figure 20: Classification of Devices

For devices we identified the following classes of device types (Figure 20):

- Tag,
- Mote,
- Mobile,
- Workstation,
- Server,
- Vehicle,
- Multimedia,
- Set of devices,
- Other device.

The *Tag* class stands for passive devices like RFID tags. The *Mote* class devices describe small resource constrained devices like those used in wireless sensor networks. *Mobile* class contains all kind of cell phones, PDA and similar devices. *Workstation* class can be used to describe all kinds of personal computers. *Server* devices are special computers capable of storing and/or computing large amount of information. *Vehicle* class shall be used to describe all kind of vehicles like remote controlled cars, autonomous robots, spacecrafts, etc. *Multimedia* covers devices like TV, HIFI, radio, etc.

An additional device class can be used to describe a *set of devices*, which can be used for groups of devices deployed as one platform. To extend this device classification class *other device* can be used.

4.7.1.3 Capabilities

Capabilities here are functional properties, which describe some abilities of the device. For capabilities we first define top-level capability classes:

- Sensor,
- Actuator,
- Human interface device (HID),
- Energy,
- Communication,
- Computing,
- Storage and
- Other capability.

These class types shall roughly describe the kind of capability. *Sensor*, *actor*, *HID*, *energy* and *storage* classes are further subclassified. *Communication* and *computing* classes are not further subclassified, because here it is more appropriate to use parameters to describe specific capabilities of these classes. For example, *communication* capability can be described by filling the following parameters: communication media, communication standard, protocol, bandwidth and so on. The *computing* capability can be described with following parameters: processor type (microprocessor, CPU, GPU, etc.), frequency, number of cores and type of instruction set used, etc.

4.7.1.4 Services

For services we identified three basic groups:

- Hardware related services,
- Software services,
- Real world services and
- Other services.

Hardware related services enable access to hardware capabilities of a node, like accessing sensors or actuators. *Software services* are all services, which are not directly dependent on some hardware capabilities. Typical software services are computing functions, virtual machines for "mobile software agents", etc. The third group of services covers services in the real world, like food delivery, transportation, etc.

4.7.1.5 Data

We decided to add a basic classification of data types, to make this basic classification complete and to allow specifying, for example, the type of data someone is requesting. We defined classification entries for the following generic data types:

- Number,
- Text,
- Date,
- Record,
- Array,
- Stream and
- Other Data.

4.7.1.6 Properties

The properties branch describes characteristics that are not capabilities of the device. These additional characteristics are often used for self-description and discovery. For example, the manufacturer and the owner of a device are typical properties. The properties are usually stored as key-value pairs.

4.7.1.7 Other

Each branch in this classification has an element "other ...". These elements are thought as points at which the classification can be extended. If, for example, the element "other device" is extended, it would mean, that all existing device types in the classification are not appropriate to describe the device being classified.

4.7.2 Classification Definition in ASN.1

Here we define an enumeration with constants for all the ontology entries. This allows to describe an entry by just one constant value, which is encodable in 7 bits. The full list is available in appendix A.2.

As long as all devices use the predefined ontology they know to what ontology entry which constant value corresponds and what the hierarchical relationship to other entries is.

A problem arises when some devices use an extended version of the ontology. When other devices get a message containing a new constant value, they can not know where this new ontology entry is placed in the classification hierarchy.

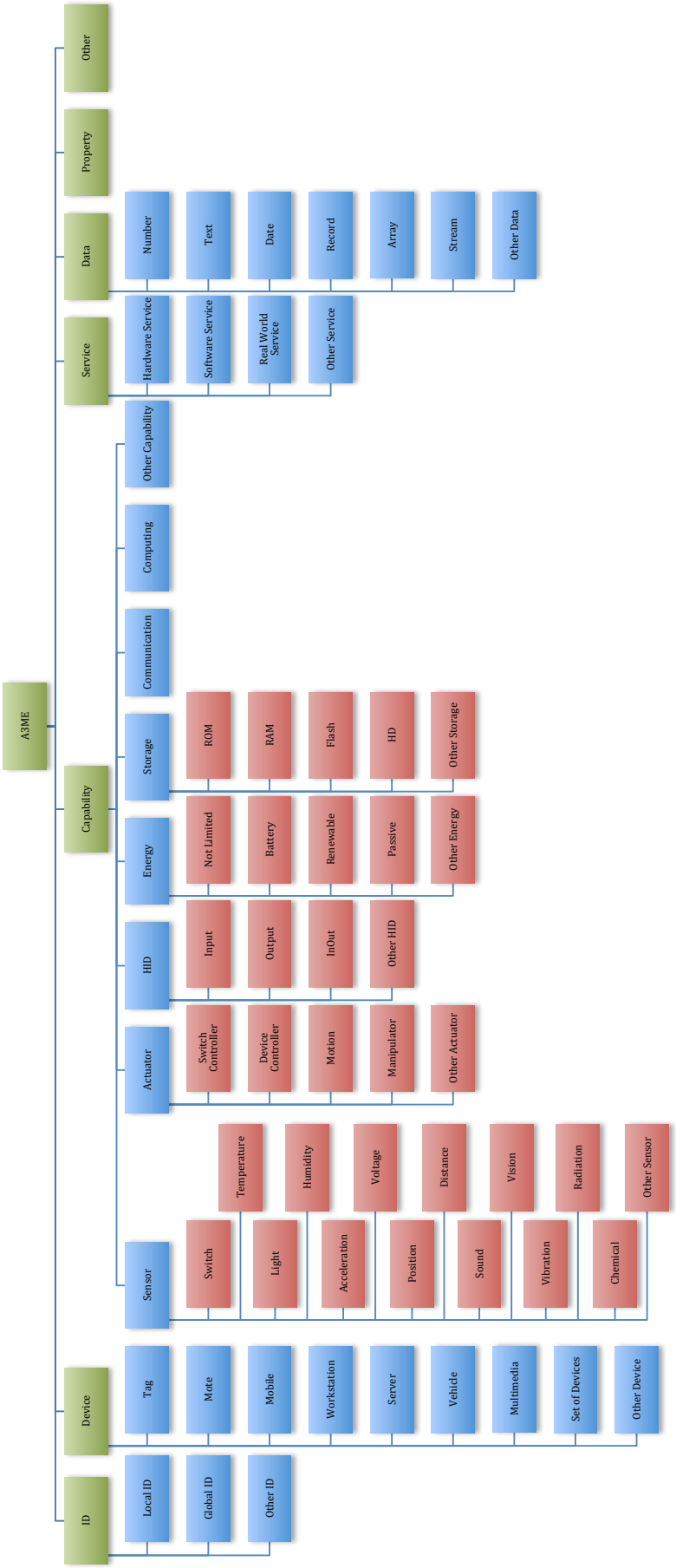


Figure 21: A3ME Predefined Extensible Ontology

The solution for this offer the *Object Identifiers (OIDs) (Section 4.7.3)*. When a device-agent receives a message containing a classification entry not known to it, it can request the OID describing the unknown classification entry from the source device-agent. The OID contains the chain of classification entries starting at the root node up to the classification entry represented by this OID. Once OID is known the device-agent can figure out at what point the classification was extended and how.

4.7.3 Assignment of Object Identifiers

An Object Identifier (OID) is syntactically an ordered list of object identifier components. "Each new node is associated with a name (a word beginning with a lowercase letter) and a number that will be used for data transfers." [36] Many standards are also identified by OIDs. Most common OIDs usually belong to the private enterprise numbers allocated by IANA under the 1.3.6.1.4.1 (iso.org.dod.internet.private.enterprise) branch.

We will reuse the existing functionality of OIDs to describe the hierarchical relationship of the A3ME ontology. As root OID we will use the OID of computer science department of the Technische Universität Darmstadt:

```
{iso(1) org(3) dod(6) internet(1) private(4) enterprise(1) 8301}
```

The classification entries of the A3ME ontology are defined as Relative OIDs, which have the root OID 1.3.6.1.4.8301.dvs(4).projects(0).a3me(0). For efficiency issues regarding message size inside the A3ME framework we will use the Relative OIDs. The relative OID for the temperature sensor is represented as:

```
{a3me(0) capability(2) sensor(1) temperature(2)}
```

And the corresponding complete OID for the temperature sensor is:

```
{1.3.6.1.4.8301.dvs(4).projects(0).a3me(0).capability(2).sensor(1).temperature(2)}
```

But those can anytime be converted to full OIDs by relating the relative OID to the root. Appendix A.5 contains the ASN.1 based definition of OIDs for the A3ME classification.

4.7.4 Classification Extension

Since a predefined classification can not be complete, meaning covering all possible types of classified entries, we designed our ontology the way it can be extended at any point.

There are two ways to extend the A3ME ontology. In both cases the best possible match from the predefined ontology (base-classification-entry) is used.

We will use the following example to demonstrate the extensibility of the ontology.

Example 2. A smart phone device needs to register two sensors not covered directly by the predefined classification:

- a rotation vector sensor and
- a magnetic field sensor.

4.7.4.1 Simple Extension

In the case of simple extension the new classification entry is represented by an instance of base-classification-entry where name and description of the new entry are set as parameters of this instance. In listing 16 we use the predefined classification entry of 'other_sensor' to define the sensor capabilities for a rotation and a magnetic sensor.

```
InfoStorage.add(A3ME_code.other_sensor,
  "Rotation Vector Sensor",
  "resolution=5.9604645E-8, minDelay=20000, maxRange=1.0, usagePower=7.03");
InfoStorage.add(A3ME_code.other_sensor,
  "AK8973 3-axis Magnetic field sensor",
  "resolution=0.0625, minDelay=16667, maxRange=2000.0, usagePower=6.8");
```

Listing 16: Simple classification extension by instantiation of an existing predefined entry.

This simple extension of the classification is especially useful on resource constrained devices.

[36] Source: http://www.itu.int/ITU-T/asn1/

4.7.4.2 Full Extension

Here the predefined ontology is extended with new entries. Meaning new constants are introduced, the ASN.1 definition of the ontology is extended using ASN.1 extension mechanisms[37] [5]. Listing 17 shows the ASN.1 definition for the full extension of the above example.

```
// A3ME constants
A3ME-code ::= ENUMERATED {
  a3me     (0),
  // here follow all the predefined entries, which are omitted in this listing
  // ...
  other    (71),
  ... ,
  [[ // here follow the two new entries
  orientation_sensor  (72),
  magneticfield_sensor  (73),
  ]]
}

// A3ME Object Identifiers to define the hierarhical relationship
// to the ontology being extended.
roid-orientation-sensor RELATIVE-OID ::= {roid-sensor orientation-sensor(14)},
roid-magneticfield-sensor RELATIVE-OID ::= {roid-sensor magneticfield-sensor(15)},
```

Listing 17: Full classification extension by extension of the ASN.1 definition of the ontology.

This extended ASN.1 definition of the ontology needs to be compiled for the target device. This means that it is required to have a corresponding ASN.1 compiler for the given programming language and given device platform. Many ASN.1 compilers are non free to use and therefore are often not shipped together with the software, but only the precompiled library.

This makes the use of full extension of the predefined ontology cumbersome. Especially on resource constrained devices this full extension of the classification usually is not possible on the devices themselves but only by replacement of the firmware.

4.7.5 Additional SSN Ontology Definitions

Additional to the A3ME classification used to describe the devices and their capabilities we can define more detailed semantical descriptions using the SSN ontology (see section 3.10.2). Overview of SSN ontology can be seen in figure 12. The markup with the SSN and eventual other ontologies is useful to enable semantical integration with other frameworks.

The **TelosB** sensor node is described as subclass of 'System' and 'Platform'. A 'System' is something that has 'Components'. So we can describe the components of the sensor node: 'Power Supply', 'LEDs' and the sensors. Each sensor has a property to measure one specific 'Feature of Interest', e.g. 'Sensor Humidity' has the property 'Humidity Measurement', which is the property of 'Humidity'.

[37] Some resource constrained devices (e.g. TelosB) don't support ASN.1 definition extensions.

```
<?xml version="1.0"?>
<!DOCTYPE Ontology [
    <!ENTITY xsd  "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY xml  "http://www.w3.org/XML/1998/namespace" >
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY rdf  "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
]>
<Ontology xmlns="http://www.w3.org/2002/07/owl#"
     xml:base  ="https://www.dvs.tu-darmstadt.de/staff/aherzog/a3me/a3me-deployment"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:xsd ="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdf ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:xml ="http://www.w3.org/XML/1998/namespace"
     ontologyIRI="https://www.dvs.tu-darmstadt.de/staff/aherzog/a3me/a3me-deployment">
    <Prefix name=""     IRI="http://www.w3.org/2002/07/owl#"/>
    <Prefix name="DUL"  IRI="http://www.loa-cnr.it/ontologies/DUL.owl#"/>
    <Prefix name="ssn"  IRI="http://purl.oclc.org/NET/ssnx/ssn#"/>
    <!-- more prefixes in the complete file -->
    <Import>http://purl.oclc.org/NET/ssnx/ssn</Import>
    <Declaration>
        <Class IRI="#Sensor_Node_TelosB"/>
    </Declaration>
    <SubClassOf>
        <Class IRI="#Sensor_Node_TelosB"/>
        <Class abbreviatedIRI="ssn:Device"/>
    </SubClassOf>
    <SubClassOf>
        <Class IRI="#Sensor_Node_TelosB"/>
        <Class abbreviatedIRI="ssn:Platform"/>
    </SubClassOf>
    <!-- rest of the description in the complete file -->
</Ontology>
<!-- Generated by the OWL API (version 3.4.2) http://owlapi.sourceforge.net -->
```

Listing 18: Excerpt of the SSN Ontology for description of the A3ME deployment.

Listing 18 shows an excerpt of the SSN ontology describing the A3ME deployment used for evaluation. Here we reference two other ontologies:

- The Semantic Sensor Network (SSN) Ontology: http://purl.oclc.org/NET/ssnx/ssn on which this ontology is based and
- The DOLCE [78] Ultra Lite (DUL) upper ontology: http://www.loa.istc.cnr.it/ontologies/DUL.owl which is used in SSN.

Figure 22 shows the visualization of the SSN ontology describing the TelosB sensor device. The solid lines indicate the Subclass relationships and the dashed lines describe semantical relations to other entities. The black framed entity boxes in the figure are entities defined in the SSN ontology. The blue and green framed boxes are entities we extended the SSN ontology with to describe the A3ME deployment used for the evaluation.

The SSN ontology allows to describe also other aspects of the (sensor) devices not described in our example ontology:

- Information about the deployment,
- The sensor precision at different conditions,
- The process of measurement,
- Description of the measurement data,
- etc.

Which aspects of the ontology are used depends on the application and usage scenario.

4.8 A3ME Message Structure

This section describes the structure of the messages, which will be transported by the different communication technologies as payload. This means the A3ME message will be put inside communication technology specific message body. The communication technology specific message might contain other information required for used protocol like sender

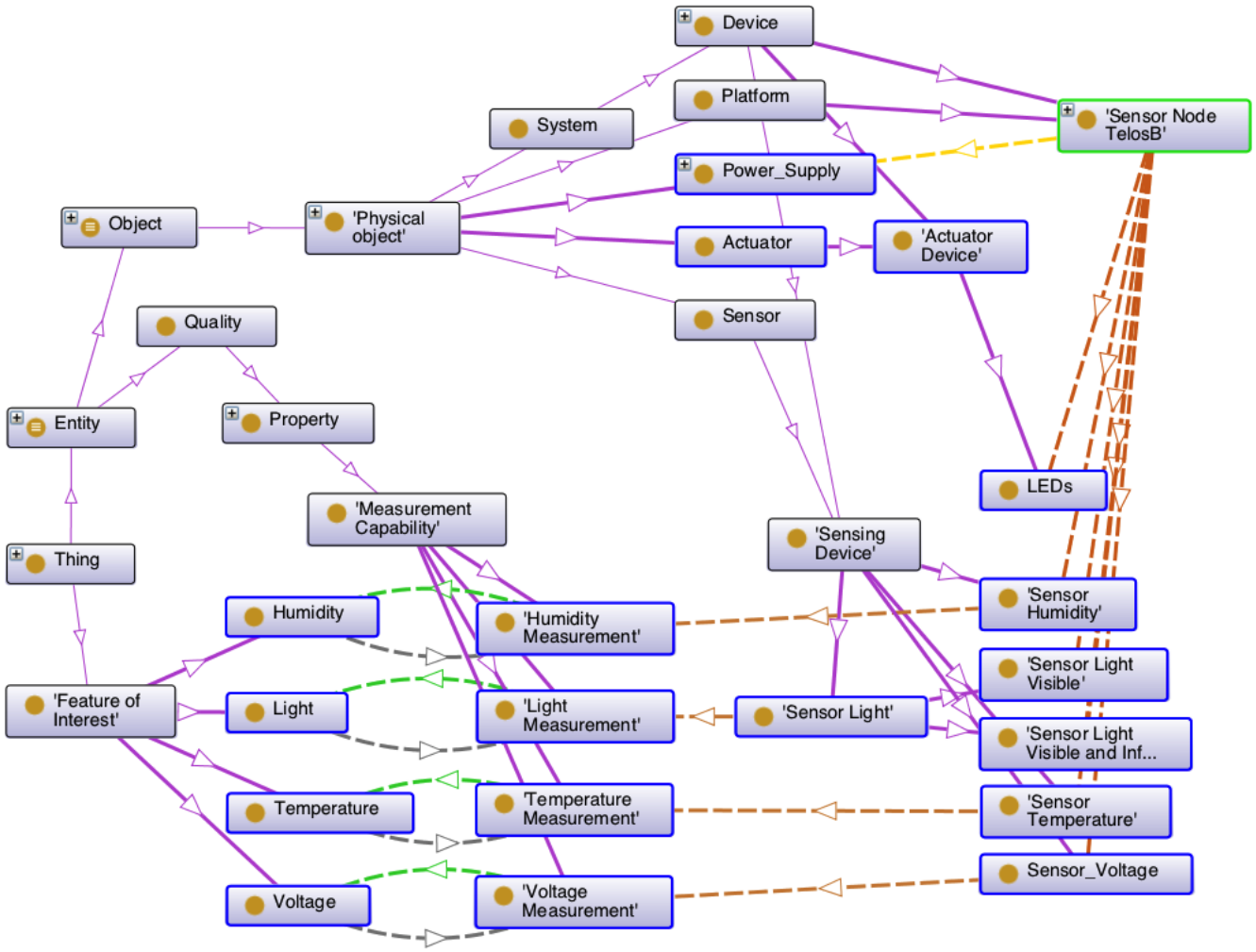


Figure 22: Semantic Sensor Network Ontology Based Description of a TelosB Sensor Node.

address, receiver address etc., which are valid for the used protocol. For example, if the used communication protocol is TCP/IP, the addresses would be IP-addresses.

The A3ME Message itself usually also contains sender and destination addresses for the message, but it also contains their IDs and potentially other kind of addresses valid for the sender and/or receiver on another communication interface.

As defined in FIPA ACL Message Structure Specification [9] the message contains parameters listed in Table 7. The only mandatory parameter is the performative.

We also introduce an additional parameter *received-from* containing the Address to enable the Device-agent to identify through which communication interface and from which node the message was received (Table 9).

Parameter	Category of Parameters	Description
received-from	Communication type and address	Communication type and corresponding address of the sender.

Table 9: Additional Message Parameters for A3ME Message

4.8.1 A3ME Message Performative

For interactions between DAs messages of predefined types are used. As types FIPA ACL performatives are used (Table 2). The idea behind performatives is that each message has some intention and each performative represents one class of intentions. Therefore each message is tagged with a performative which defines its purpose e.g. request. This makes it possible to know the purpose of the message just by the type of the message without knowing the content.

The message types we currently use in the A3ME framework are the request, inform, refuse, cancel and not-understood performatives.

- The *request performative* means the message contains some kind of request and will trigger an answer from the receiver(s).
- The *inform performative* is used to answer a query or to inform others proactively either periodically, on start up or on some status change.
- The *refuse performative* is used to decline a request. The reason can be attached to the message as text.
- The *cancel performative* is used to cancel some continuous query or service execution. What has to be cancelled is specified by the Query-ID used to trigger the query or service.
- The *not-understood performative* is used to tell the receiver that the message was not understood by the receiver. The reason can be attached to the message as text.

4.8.2 A3ME Message Content

The content of the messages for interactions between the devices should be easily machine readable, meaning it should be formulated in a well defined formal language. Furthermore the used format should allow to formulate the messages dynamically, meaning that the message structure should be dynamic and not static. There is a variety of languages which could be used for this.

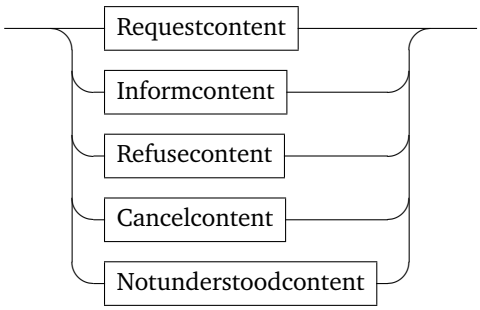
XML (section 3.8.1), SOAP (based on XML, section 3.8.3) and other XML based formats are often used to describe contents exchanged between conventional computers. All this formats allow to define schemata, to which the data defined for that format must correspond. This also can automatically be validated. The disadvantage of all XML based formats is the significantly increased size of the messages and the construction and parsing of these messages requires a significant amount of computing and storage resources.

Light-weighted formats like JSON (section 3.8.4) and YAML (section 3.8.7) are more efficient in terms of message size, but lack the possibility to be verified against a specified schema.

ASN.1 (Abstract Syntax Notation One, section 3.8.5) combines the advantages of compact messages and validation possibility. Furthermore if used in combination with ASN.1 Packed Encoding Rules (PER, section 3.8.5.2) the message size can be reduced even more.

The message content is specific to the message type and is represented using the A3ME Content Definition (See section 4.10).

Messagecontent



The different content types are described in more detail in section 4.10.

4.9 Device Interaction Primitives

The main goal of this work is to enable ad-hoc interactions between electronic devices. With interactions here we mean the exchange of electronic messages. We don't consider other forms of interactions like physical interactions between devices in the first place. But these interactions might trigger or be a consequence of some physical interactions of the individual devices with the real world.

All communication-based interactions between electronic devices happen through exchange of messages. According to the speech act theory formalized in FIPA communicative acts [10] each message implies some type of action (section 3.5.6). FIPA defines 22 different types of actions – performatives (Table 2). In our framework we reuse the performatives defined by FIPA and assign all messages with a message type – performative. By assigning one of these performatives to each message it is often possible to assign the right operation to deal with the message just by knowing its performative often even without knowing the content.

Most interactions between devices can be classified either as information exchange or as service invocation or as both. In A3ME we consider both types.

In A3ME the interactions are based on classes of capabilities defined in the A3ME Classification. This allows to discover and to interact with other devices independent of their status of being already known or not.

Here a minimal set of interactions is defined, which are enabled through A3ME between different devices through device-agents. These protocols specify the flow of message types to be exchanged for each type of interaction.

4.9.1 Inform Interaction

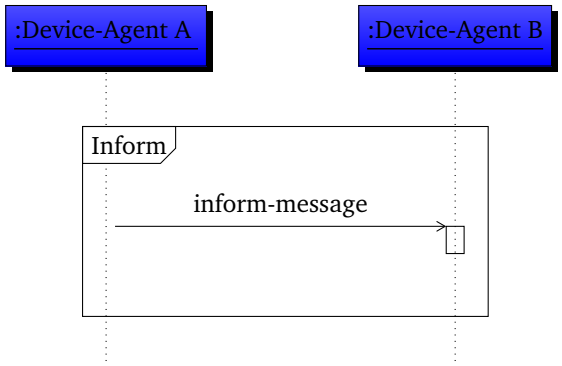


Figure 23: UML Sequence Diagram of an Inform Interaction.

The Inform Interaction Protocol (Figure 23) is used to introduce a device-agent to others. This is usually done when a device is switched on, enters a new area while moving or periodically. In this interaction just one Inform message is send and does not require other device-agents to react to it.

4.9.2 Request Interaction

The Request Interaction Protocol is the protocol used for all kind of requests. If the receiving device-agent is named directly as recipients, it has to follow the request interaction protocol described in Figure 24. Requests that are addressed to all (it is the case if no recipient is specified) or to groups of devices do not require any reaction (agree or refuse) from the recipient. This way we avoid flooding of the network with unnecessary messages.

If the request is not addressed for the receiving device-agent (directly, via group addressing or if send to all), the message only might need to be forwarded. Whether the message is forwarded depends on:

- the routing protocols used,
- whether the forwarding limitation is defined and exceeded,
- local policies and
- device status.

4.9.3 Service Call Interaction

The Service Call Interaction Protocol is the protocol used to call services on device-agents. This Interaction protocol is similar to the Request Interaction (Section 4.9.3), with the difference that instead of sending an answer to the requester a local service is called.

The UML sequence diagram in Figure 25 shows the interaction:

1. Device-agent A sends a request-message with a service-call request to one or multiple device-agents.
2. Each receiving device-agent (represented in the diagram as Device-Agent B) deals with the request as follows:

```
if the message can not be understood:
    then reply with not-understood message,
    else If the request is addressed to this device-agent and applicable:
        then if the local policies prevent the requested service call, send a refuse-message.
        else execute the call.
```

3. Device-agent A can send a cancel-message to cancel a service call.

4.10 A3ME Content Representation in ASN.1

As described in section 4.1.1 it is reasonable to have a neutral representation when interconnecting different communication technologies. During the investigation for a suitable existing technology capable to do this we first had a closer look at the FIPA ACL BE (section 3.8.6) but then we identified the ASN.1 standard in combination with ASN.1 Packed Encoding Rules (PER) to be the perfect solution to encode and decode messages in a MME. The ASN.1 standard was originally defined in 1984 and is used for over 30 years in wide range of technologies.

We used the ASN.1 syntax notation to describe the contents we need for description, storage and exchange of information in the A3ME framework. The following sub sections describe the definition of the different content types in more detail. The ASN.1 definitions used in A3ME are divided into 3 parts: Message definition (section A.3), Content definition (section A.4), A3ME ontology definition (section A.2).

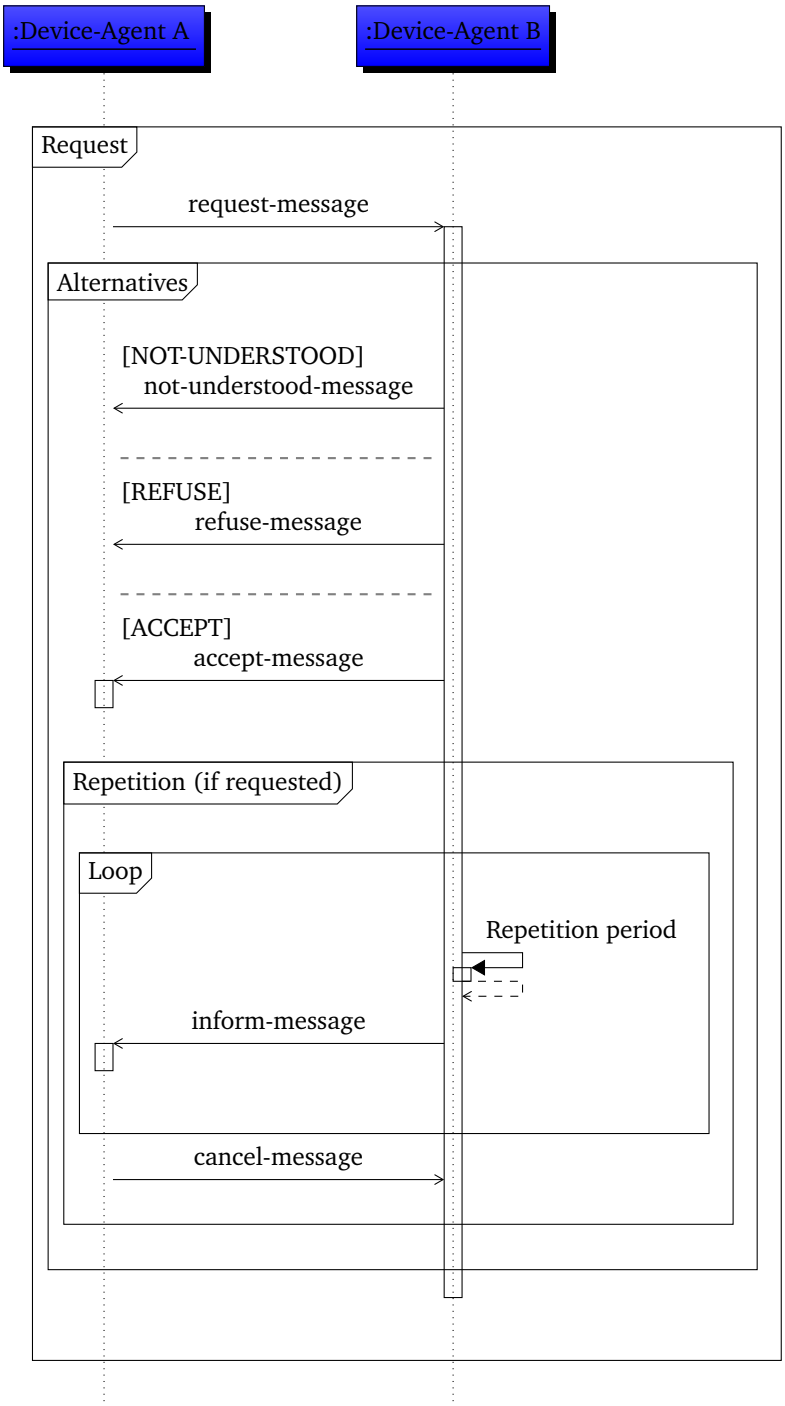


Figure 24: UML Sequence Diagram of a Request Data Interaction.

4.10.1 Character encoding

For character encoding we first considered using of UTF-8 instead of ASCII. This way we would not limit the strings to ASCII characters only and as long as only ASCII characters would be used the encoded byte value also wouldn't take more space as if ASCII would be used. But when implementing it on resource constrained devices it was recognized that those often don't support the UTF-8 character encoding. This means it would be necessary to implement it or respectively add the necessary libraries, which on the other hand would increase the code size. Therefore we went back to use US-ASCII character encoding.

4.10.2 Common Elements

In this section we define the common content elements used in different message types.

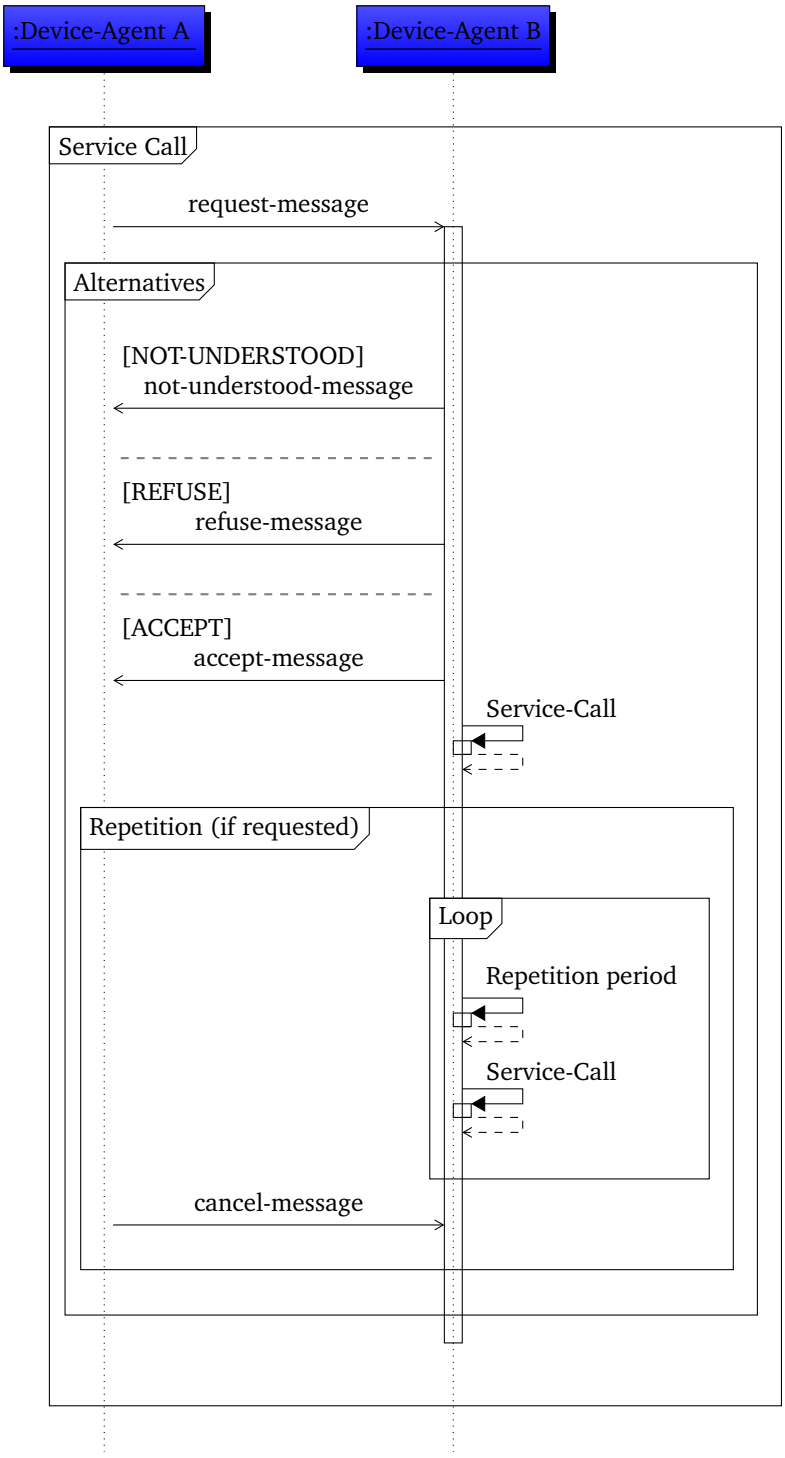


Figure 25: UML Sequence Diagram of a Request Service Call Interaction.

4.10.2.1 Infotype

We defined a new specifier *Infotype* to describe the different types of information. This specifier can be used to specify the information in requests and replies. We identified the following *Infotypes* which we use in this thesis:

- Type-Code: encoding number from the ontology.
- Type-Name: human readable name for the type defined in the ontology.
- Name: human readable name of the object.
- Description: human readable description of the object.
- ID: ID for the object.
- Data: data value(s) for the object (e.g. sensor readings).

- M2M_Description: machine readable description (e.g. WSDL document).

In section 6.2 we define various queries which demonstrate the use of the *Infotype*.

```
Infotype     ::= ENUMERATED {
    type-code,       --encoding number from the A3ME ontology.
    type-name,       --type name from the A3ME ontology.
    name,            --human readable Name of the object.
    description,     --human readable descr. of the object.
    id,              --ID for the object
    data,            --data value(s) (e.g. sensor readings)
    m2m-description,--machine readable descr (e.g. WSDL doc.)
    ...              --possible extensions
}
```

4.10.2.2 Data-descriptor

A *Data-descriptor* is a pair of *A3ME-code* and the *Infotype* and describes one data item in a request or answer message. A *Data-descriptors* element is a sequence of *Data-descriptor* elements.

```
Data-descriptors     ::= SEQUENCE (SIZE(0..1023)) OF Data-descriptor

Data-descriptor      ::= SEQUENCE {
    a3me-code A3ME-code,
    infotype Infotype
}
```

4.10.2.3 A3ME-code

The ontology codes (A.1) are defined as ENUMERATED data type in ASN.1 (see appendix A.2). This way they can be encoded very compactly.

4.10.2.4 DaID

DaID represents the device-agent ID and contains the name and the addresses of the device-agent.

```
DaID             ::= SEQUENCE {
    name             StringType,
    addresses Addresses (SIZE(1..64)) OPTIONAL
}
```

4.10.2.5 Address and Addresses

An *Address* element contains the string representation of a communication address and the corresponding address type. The *Addresses* element is a sequence of *Address* elements.

```
Addresses        ::= SEQUENCE (SIZE (0..1023)) OF Address

Address          ::= SEQUENCE {
    address-type     StringType (SIZE(1..16)),
    address              StringType (SIZE(1..256))
}
```

4.10.2.6 Operator

The *Operator* element is an enumeration of comparison operators: equals, greater, greater-equal, smaller, smaller-equal.

```
Operator     ::= ENUMERATED {
    equals,
    greater,
    greater-equal,
    smaller,
    smaller-equal,
    ...              --possible extensions
}
```

4.10.2.7 Time-value

The *Time-value* contains an integer and a time unit value.

```
Time-value   ::= SEQUENCE {
    number INTEGER(0..4294967295),
    time-unit Time-unit
}
```

4.10.2.8 Time-unit

The *Time-unit* is an enumeration of different time units, which can be used in requests.

```
Time-unit    ::= ENUMERATED {
    nanosecond,
    millisecond,
    second,
    minute,
    hour,
    day,
    week,
    year,
    ...              --possible extensions
}
```

4.10.2.9 Distance-unit

The *Distance-unit* is an enumeration of different distance units, which can be used in requests.

```
Distance-unit    ::= ENUMERATED {
    hop,
    meter,
    kilometer,
    ...              --possible extensions
}
```

4.10.2.10 Resultset

The *Resultset* is used to answer information requests and represents a data table with a schema and data rows.

```
Resultset    ::= SEQUENCE {
    schema  Data-descriptors,
    rows SEQUENCE (SIZE (0..65535)) OF Data-record
}
```

4.10.2.11 Record

The *Record* is a sequence of *Data-item* elements.

```
Record       ::= SEQUENCE (SIZE (0..1023)) OF Data-item
```

4.10.2.12 Data-record

The *Data-record* is a sequence of *Data* elements.

```
Data-record      ::= SEQUENCE (SIZE (0..1023)) OF Data
```

4.10.2.13 Data-item

The *Data-item* is a combination of *Data-descriptor* and the corresponding *Data*.

```
Data-item    ::= SEQUENCE {
    data-descriptor Data-descriptor OPTIONAL,
    data Data OPTIONAL
}
```

4.10.2.14 Data

The *Data* element contains one of the standard data elements.

```
Data ::= CHOICE {
    integer-data INTEGER(-2147483648..2147483647),
    real-data    REALType, -- REAL type not supported by oss tool for JavaME
    boolean-data BOOLEAN,
```

```
    string-data StringType(SIZE(0..65000)),
    -- GeneralizedTime support is disabled due to encoder size constraints on small devices therefore
        useng a string version
    date-data    GeneralizedTimeString,
    time-data Time-data,
    byte-data OCTET STRING(SIZE (0..65000)),
    bit-string BIT STRING(SIZE (0..65000)),
    null NULL,
    record-data Record,
    --key-value-pair         Key-value-pair,
    ...
}
```

4.10.2.15 REALType

The *REALType* is an data element to represent floating point numbers. It was introduced here to replace the ASN.1 build-in REAL data type.

```
REALType            ::= SEQUENCE { /* Definition from ASN.1-X.680 */
    mantissa INTEGER(-2147483648..2147483647),
    base INTEGER (2|10),
    exponent INTEGER(-46340..46340)
    -- The associated mathematical real number is "mantissa"
    -- multiplied by "base" raised to the power "exponent"
}
```

4.10.2.16 Time-data

The *Time-data* is a sequence of *Time-values* and is used to represent time and date.

```
Time-data    ::= SEQUENCE(SIZE (0..10)) OF Time-value
```

4.10.3 A3ME Messages

The definitions cover the following message elements:

- Message performatives,
- Message parameter types,
- Message content types.

4.10.3.1 Message performatives

As described in section 4.8.1 each A3ME message has message type – performative, which determines the purpose of the message.

The 22 message performatives[10] are defined as ENUMERATED ASN.1 data type in listing 19.

```
Performative ::= ENUMERATED {
    /** FIPA performative constants **/
    accept-proposal ,--(0),
    agree           ,--(1),
    cancel          ,--(2),
    cfp             ,--(3),
    confirm         ,--(4),
    disconfirm      ,--(5),
    failure         ,--(6),
    inform          ,--(7),
    inform-if       ,--(8),
    inform-ref      ,--(9),
    not-understood  ,--(10),
    propose         ,--(11),
    query-if        ,--(12),
    query-ref       ,--(13),
    refuse          ,--(14),
    reject-proposal ,--(15),
    request         ,--(16),
    request-when    ,--(17),
    request-whenever,--(18),
    subscribe       ,--(19),
    proxy           ,--(20),
    propagate       ,--(21),
```

```
    unknown          --(-1) represented as 22 here
}
```

Listing 19: Message performatives

4.10.3.2 Message parameter types

The message parameters are defined according to FIPA Message Specification [9] (listing 20). The only mandatory parameter is the *performative*. All other parameters are optional.

```
Message ::= SEQUENCE {
        performative        Performative,
        sender              DaID          OPTIONAL, --Denotes the identity of the sender of the message
        receiver            Addresses     OPTIONAL,
        reply-to            DaID              OPTIONAL,
        content             Message-content OPTIONAL,
        language            Language      OPTIONAL,
        encoding            Encoding      OPTIONAL,
        ontology            Ontology      OPTIONAL,
        protocol            Protocol      OPTIONAL,
        conversation-id     ConversationID  OPTIONAL,
        reply-with          ConversationID  OPTIONAL,
        in-reply-to         ConversationID  OPTIONAL,
        reply-by            GeneralizedTime OPTIONAL,
        received-from       Address       OPTIONAL
}
```

Listing 20: Parameters

4.10.3.3 Message Content Types

This section describes the syntax and semantic for the A3ME messages. Listing 21 shows the definition of different message content types currently realized. The different types are described in more detail later in this section.

```
Message-content ::= CHOICE {
    request-content          Request-content,
    inform-content           Inform-content,
    refuse-content           Refuse-content,
    not-understood-content   Not-understood-content,
    cancel-content           Cancel-content,
    encrypted-content        Encrypted-content,
    ...
}
```

Listing 21: Message Content Types

These content definitions are explained in more detail in the following sections.

4.10.4 Request Message Content

With a request message other devices can be asked to deliver information or execute a service. The request can have conditions, which have to be satisfied for the requested information and are specified in the WHERE clause. The information can be requested to be delivered periodically (PERIOD) for a specified time DURATION. To limit how far the request has to be forwarded in the network, a maximum RANGE can be defined. Before forwarding the request each node subtracts his average radio transmission range from the range distance in the forwarded request.

Request content can be defined directly using A3ME Content Definition in ASN.1 notation or using the SQL like A3ME Query Language (A3ME-QL) (See section 4.11).

```
Request-content ::= SEQUENCE {
    what     What,
    from     From-clause      OPTIONAL,
    where    Condition-clause     OPTIONAL,
    period   Period-clause        OPTIONAL,
    range    Range-clause         OPTIONAL
}
```

4.10.4.1 What

The *What* specifies the data or the service which is requested.

```
What ::= CHOICE {
    data-columns Data-descriptors,
    service-call Service-call
}
```

4.10.4.2 Service-call

The *Service-call* specifies the service, the command to execute and parameters if required.

```
Service-call ::= SEQUENCE {
    service CHOICE {
        id INTEGER(0..65535),
        capability-code A3ME-code
        --service-code A3ME-code
    },
    command INTEGER(0..1023),
    parameters Record OPTIONAL
}
```

4.10.4.3 From-clause

The *From-clause* is a sequence of *DaID*s addressed by the request. If the *From-clause* is omitted, FROM ALL is expected as default value and means that the request is addressed to all.

```
From-clause      ::= SEQUENCE (SIZE (0..1023)) OF DaID --[FROM (ALL / daID *[, daID]) ]
```

4.10.4.4 Condition-clause

The *Condition-clause* is a sequence of *Condition* elements and represents filter conditions which all must be fulfilled for the query.

```
Condition-clause    ::= SEQUENCE (SIZE (0..1023)) OF Condition --[WHERE condition *[AND condition]]
```

4.10.4.5 Condition

The *Condition* represents a filter condition for the query.

```
Condition         ::= CHOICE {
    a3me-code           A3ME-code,          --existence condition of given a3me-code on DA
    is-for-condition    Is-for-condition,   -- IS-FOR data-descriptor a3me-code e.g. service for temp
    operator-condition  Operator-condition  -- operator data-descriptor string
}
```

4.10.4.6 Is-for-condition

The *Is-for-condition* asks for data where the *Data-Descriptor* in the what part is related to the *A3ME-code* specified here.

```
Is-for-condition    ::= SEQUENCE {
    data-descriptor Data-descriptor,
    a3me-code A3ME-code
}
```

4.10.4.7 Operator-condition

The *Operator-condition* compares one of the *Data-descriptor*s with specified *Data-item*.

```
Operator-condition  ::= SEQUENCE {
    operator        Operator,
    data-descriptor Data-descriptor,
    parameter       Data-item
}
```

4.10.4.8 Period-clause

The *Period-clause* specifies the repetition interval and duration of the query to be executed.

```
Period-clause   ::= SEQUENCE {
    period Time-value,
    duration Time-value OPTIONAL
}
```

4.10.4.9 Range-clause

The *Range-clause* can be used to limit the distribution of the query by range.

```
Range-clause     ::= SEQUENCE {
    number INTEGER(0..4294967295),
    distance-unit Distance-unit
}
```

4.10.5 Inform Message Content

Inform messages are used for notifications and to answer request. Depending on the request it can be a single value, a series of periodical answers or a set of answers.

```
Inform-content   ::= SEQUENCE {
    sequence-number INTEGER(0..4294967295) OPTIONAL,
    resultset Resultset
}
```

4.10.6 Refuse Message Content

Refuse is used to reject a received request for any reason. The reason can be given as string.

```
Refuse-content   ::= SEQUENCE {      --REFUSE requestID [string]
    reason StringType(SIZE(0..1023)) OPTIONAL
}
```

4.10.7 Cancel Message Content

Cancel is used to chancel a request for any reason. The reason can be given as string.

```
Cancel-content   ::= SEQUENCE {
    reason StringType(SIZE(0..1023)) OPTIONAL
}
```

4.10.8 Not-understood Message Content

This message can be sent as an answer whenever the receiving node cannot deal with the request. Reasons for this can be:

- the message received is not supported,
- message was corrupted,
- the query language used is not supported,
- etc.

```
Not-understood-content   ::= SEQUENCE {
    reason StringType(SIZE(0..1023)) OPTIONAL
}
```

4.10.9 Encrypted Message Content

Encrypted message content allows to encrypt any data and store it as byte stream inside this element.

```
Encrypted-content    ::= SEQUENCE {
    ciphertext       OCTET STRING(SIZE(0..65000)),
    algorithm        StringType(SIZE(0..127)) OPTIONAL,
    encrypted-for    StringType(SIZE(0..127)) OPTIONAL
}
```

4.10.10 Extension of the Definitions

The A3ME ASN.1 definitions can be extended at multiple places. This places are marked in ASN.1 with "...". See also section 4.7.4.

4.11 A3ME Query Language (A3ME-QL)

The content of the A3ME messages is build following the A3ME Content Definition (Section 4.10). Defining queries directly in the ASN.1 syntax is quite cumbersome for humans. Therefore we defined a SQL like query language (A3ME-QL) to formulate requests.

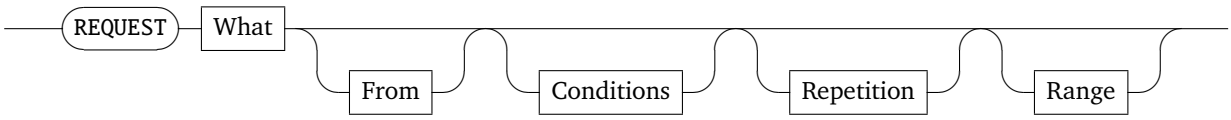
The grammar is defined in Extended Backus-Naur Form (EBNF). Complete grammar definition is listed in appendix A.6.

Queries formulated in A3ME-QL are automatically translated into the ASN.1 syntax. The ASN.1 representation is used to build internal structures and can be efficiently encoded for transmission.

4.11.1 Request-content

```
Request-content ::= ( "REQUEST"
        What
        (From-clause         )?
        (Condition-clause    )?
        (Period-clause       )?
        (Range-clause        )?
)
```

Requestcontent



Request-content is introduced by the keyword *REQUEST* followed by the *What* term. Additionally the following optional terms can be present:

- *From*: devices or addressable group of devices.
- *Conditions*: set of conditions to apply.
- *Repetition*: how often and how long to repeat the query.
- *Range*: limits the dissemination of the query via distance.

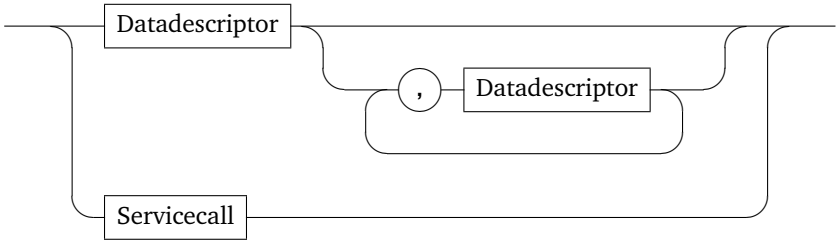
The railroad diagram[38] shows the corresponding part of the grammar.

4.11.2 What

```
What ::= (
        Data-descriptors
    |   Service-call
)

Data-descriptors    ::= Data-descriptor (',' Data-descriptor)*
```

What



The *What* term can be either:

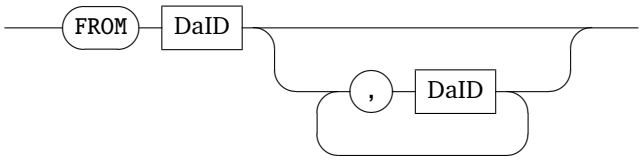
- A set of *Datadescriptor* separated by a comma or
- A *Servicecall* specifying the service, a command and an optional list of parameters.

[38] The railroad diagram generator (LaTeX rail 1.3.0) does not support hyphens therefore some names do not contain a hyphen and differ thereby from the corresponding grammar definition.

4.11.3 From-Clause

```
From-clause      ::= "FROM" DaID (',' DaID)*
```

From

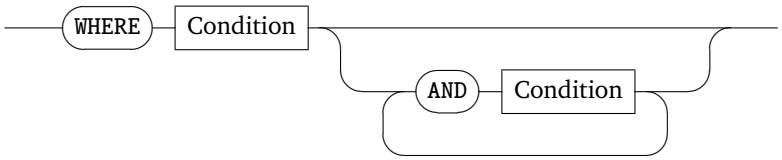


The *From* term contains a list of device-agent IDs (DaIDs), which are either individual DaIDs or group DaIDs describing whole group of devices.

4.11.4 Condition-Clause

```
Condition-clause    ::= "WHERE" Condition ("AND" Condition)*
```

Conditions

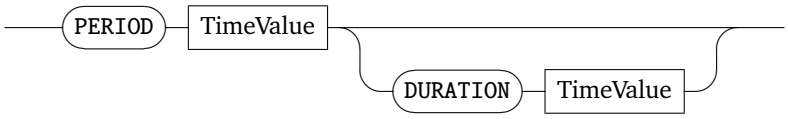


The *Conditions* term contains one or more *Condition* terms. The keyword *WHERE* marks the beginning of the *Where* part. Further conditions can be added with the *AND* keyword.

4.11.5 Repetition-Clause

```
Period-clause   ::= (
    "period" Time-value
    ("duration" Time-value)?
)
```

Repetition



The *Repetition* term is introduced by the keyword *Period* and might be followed by the optional *Duration* term. *Period* defines the time interval for the repetition of the query. The *Duration* describes how long this repetition has to be executed (See example 4).

4.11.6 Range-Clause

```
Range-clause    ::= (
    "RANGE" Digits Distance-unit
)
```

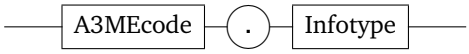
Range

The *Range* term allows to limit the dissemination of the query by distance from the requester. The term is introduced by the keyword *RANGE* followed by the *Distance* term.

4.11.7 Datadescriptor

```
Data-descriptor     ::= A3ME-code '.' Infotype
```

Datadescriptor

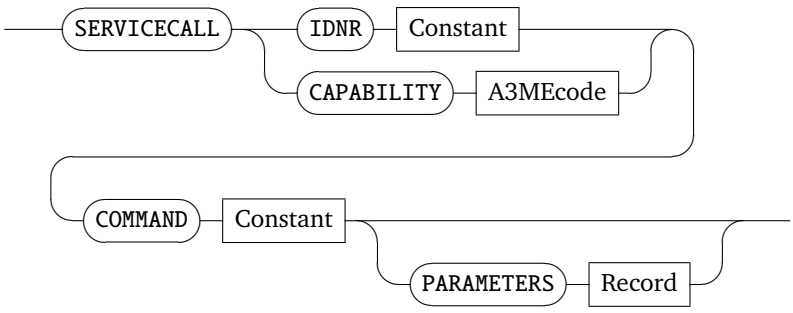


Datadescriptor describes a data item. It is defined as combination of an *Queryobject* and an *Infotype* separated by a dot. This kind of representation is selected to follow the syntax used to access object parameters, which most users intuitively understand. For example to request the temperature values *temperature.data* would be requested and to ask for different sensor type codes, we can request *sensor.code*.

4.11.8 Servicecall

```
Service-call ::= (
    "service-call" (
            "ID" Digits
        |   "CAPABILITY" A3ME-code
    )
    "command" Digits
    ("parameters" Record)?
)
```

Servicecall

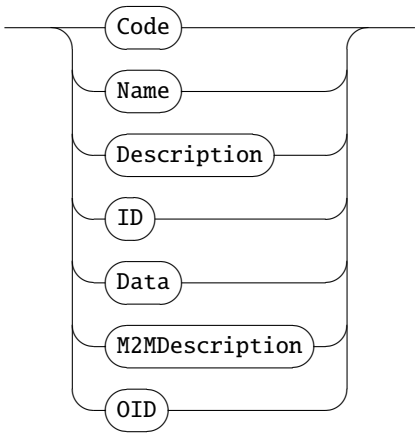


The *Servicecall* term describes the service to be called, a command and an optional set of parameters. The *Servicecall* term is introduced by the keyword *SERVICECALL* followed by the service description. A service can be specified by an ID explicitly or implicitly by the capability to which the service belongs.

4.11.9 Infotype

```
Infotype    ::= (
        "type-code"
    |   "type-name"
    |   "name"
    |   "description"
    |   "id"
    |   "data"
    |   "m2m-description"
    |   "oid"
)
```

Infotype



Infotype (section 4.10.2.1) defines what is requested and can have one of the following values:

- Code: Encoding number from the ontology.
- Name: Human readable Name of the object.
- Description: Human readable description of the object.
- ID: device ID for the object.
- Data: Data Value(s) for the object (e.g. sensor readings).
- M2M_Description: machine readable description (e.g. WSDL document).
- OID: Object Identifier describes the classification path. It is used to deal with extended ontologies to figure out how and where the ontology has been extended.

4.11.10 A3ME-code

A3ME-code is a code number from the A3ME classification (or its extension) defined in sections 4.7 and A.2. In A3ME-QL the human readable names of the classification are used, e.g. *device* or *software-service*. The *A3ME-codes* are usually used in combination with the *Infotype* separated by a dot, e.g. *device.name* or *software-service.m2m-description* (see section 4.11.7).

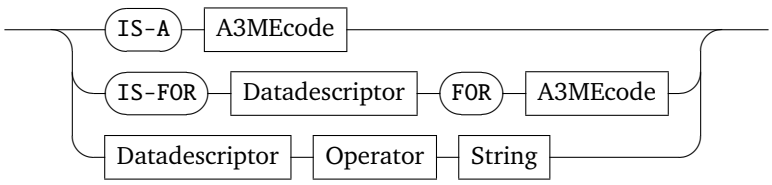
4.11.11 Condition

```
Condition       ::= (
        "is-a"      A3ME-code
    |   "is-for"    Is-for-condition
    |   Operator-condition
)

Is-for-condition    ::= (
    Data-descriptor "FOR" A3ME-code
)

Operator-condition  ::= (
    Data-descriptor Operator Data-item
)
```

Condition



Currently the A3ME framework offers the following condition types:

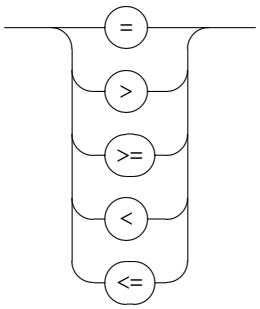
- IS-A condition: is represented as the keyword *IS-A* and an *A3MEcode*, which is evaluated as existence condition of the corresponding a3me-code locally on the device.

- IS-FOR condition: requires a data-descriptor to be related to a capability e.g. service for temperature-sensor.
- Operator condition: compares the Data-descriptor with given value. The set of available operators can be extended later.

4.11.12 Operator

```
Operator     ::= (
        "="      /*equals*/
    |   ">"      /*greater*/
    |   ">="     /*greater-equal*/
    |   "<"      /*smaller*/
    |   "<="     /*smaller-equal*/
)
```

Operator



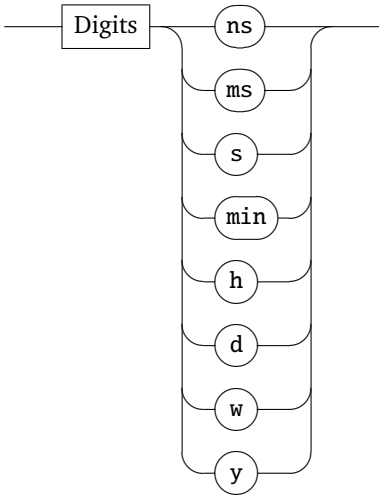
The operator can be equals, greater, greater-equal, smaller or smaller-equal. The set of available operators can be extended later.

4.11.13 Time-value

```
Time-value  ::= (
    Digits Time-unit
)

Time-unit   ::= (
        "ns"     /*nanosecond*/
    |   "ms"     /*millisecond*/
    |   "s"      /*second*/
    |   "min"    /*minute*/
    |   "h"      /*hour*/
    |   "d"      /*day*/
    |   "w"      /*week*/
    |   "y"      /*year*/
)
```

Timevalue



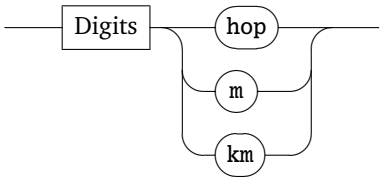
Time-value is a *long integer* number concluded with a time unit

- ns – nanoseconds,
- ms – milliseconds,
- s – seconds,
- min – minutes,
- h – hours,
- d – days,
- w - weeks,
- y - years.

4.11.14 Distance

```
Distance-unit    ::= (
        "hop"    /*hop*/
    |   "m"      /*meter*/
    |   "km"     /*kilometer*/
)
```

Distance



Distance is an integer number concluded with a unit of measurement

- hop – hops,
- m – meters,
- km – kilometers.

4.11.15 Examples

Example 3. Request all device names, which are less than 500 meter away.

In A3ME-QL:

```
REQUEST device.name
FROM ALL
RANGE 500m
```

Example 4. Request the descriptions for all temperature sensors.

In A3ME-QL:

```
REQUEST sensor.description
WHERE IS-A temperature
```

Example 5. Request ids from temperature related services.

In A3ME-QL:

```
REQUEST service.id
WHERE service.id FOR temperature
```

Example 6. Request the service with id '123'.

In A3ME-QL:

```
REQUEST service
WHERE service.id = '123'
```

Example 7. Request the temperature readings every 5 minutes for the next 10 days.

In A3ME-QL:

```
REQUEST temperature.data
PERIOD 5m DURATION 10d
```

4.12 Translation of A3ME-QL Queries into ASN.1

Queries defined with A3ME-QL are translated into ASN.1 syntax for A3ME content (Section 4.10). For this we can use grammar parser/compiler tools for the specified programming language, e.g. JavaCC for Java.

The grammar parser can run syntactical and semantical checks on the entered queries. And once those succeed the corresponding compiler can translate the queries into ASN.1 notation for A3ME content definition.

To enable the grammar parser/compiler to do this, we defined:

- Tokens (keywords, data-types, etc.),
- Language grammar in a syntax accepted by the tool (BNF),
- Rules for translation of each part of the grammar into ASN.1.

The appendix A.7 shows the definitions required by JavaCC tool to translate A3ME-QL queries into the ASN.1 notation of the A3ME content.

4.13 Message Content Encoding/Decoding

For transmission of the messages on top of any communication technology those must be encoded to a byte-stream and on reception at the destination decoded back from byte-stream to machine readable message. Often the messages are just sent as text (e.g. all XML based formats) or serialized for (e.g. Java object serialization). These serialized data can be additionally compressed to reduce the size of the resulting message.

For encoding of ASN.1 (Abstract Syntax Notation One, 3.8.5) data structures, which we use in our framework, we identified the ASN.1 unaligned packed encoding rules (PER) [6] defined in 2002 as most appropriate to be used in our framework. PER uses the knowledge of the data structure defined in the ASN.1 definition of the data to encode it in a very efficient way with respect to the byte length of the resulting byte stream. This reduces the amount of data which needs to be transmitted and received considerably.

For exchanging messages among heterogeneous devices the encoding and decoding of the messages is defined independent of the programming language and the communication technology. There exist many ASN.1 encoder/decoder implementations, but only a few of those support unaligned PER for different programming languages (in our case we needed it for Java and C). We decided to use for our prototypical implementation the tools from OSS Nokalva, Inc.[39] The OSS Nokalva provided us a research license for their ASN.1 tools for Java, Java ME and C. Additionally they developed an striped down ASN.1 encoder/decoder version for the Contiki OS for the TelosB platform, to be used for our prototype implementation.

4.14 Local API

The local API is the device local interface, which allows applications running on the same device to use functionalities offered through A3ME.

One typical application connected through the local API is a graphical user interface. It can present the user the information about the current device and about other devices, about which information is available. The user also might be offered the possibility to trigger queries to collect/update the information about other devices. These queries can be either predefined or can be entered/edited by the user.

Another possibility to use the local API is by other applications, which can use the A3ME framework to collect/query information about other devices or to exchange information with other devices trough the A3ME framework.

The A3ME Local API Java interface (Listing 22) offers two methods :

[39] OSS Nokalva, Inc. company web page: http://www.oss.com.

- send(..) to send messages through the A3ME framework and
- listen(..) to register an listener for incoming messages.

```
/*
 * A3ME framework
 * author: Arthur Herzog
 */

package a3me;

import a3me.asn.a3memessage.Message;
import java.io.IOException;

/**
 *
 * @author aherzog
 */
public interface A3ME_API extends IDeviceAgent {

  /**
   *
   * @param msg
   * @return
   */
  public boolean send(Message msg);

  /**
   *
   * @param ml
   * @throws IOException
   */
  public void listen(IMessageListener ml) throws IOException;

}
```

Listing 22: Java A3ME_API interface

Additionally the A3ME-API interface inherits the interface IDeviceAgent.java (Listing 23) and offers therefore access to the public methods of the device-agent realization to access:

- own DaID,
- Message Handler,
- Communication Component,
- Info-Store,
- Service Directory,
- Dynamic Info.

5 Prototypical implementation

To show the functionality of the A3ME framework we implemented prototypes for different platforms. In this section we describe the different implementations and in section 6 we describe different experiments we run with the prototypes.

5.1 Core Device-Agent Interface Implementation in Java 1.4

On many platforms we use different kind of Java Virtual Machine:

- Workstation: Java Standard Edition 6.
- Sun SPOT: Java Micro Edition (JavaME) with Connected Limited Device Configuration (CLDC-1.1).
- ROS: Robot Operating System module for A3ME implemented in Java using Java Native Interface (JNI).
- Android smartphone: Dalvik virtual machine which provides most of the functionality available in the core libraries of the Java programming language.

Therefore we implemented the parts which are common for the different platforms as a Core Device-Agent Interface Implementation in Java 1.4. So it can be reused on the different Java supporting platforms.

5.1.1 Interfaces

For the basic components of the device-agent realization we defined Java interfaces, which allows us to use different implementing classes for this components.

5.1.1.1 Device-Agent Interface

The device-agent interface (Listing 23) defines all the methods to access the different main components and the Device-agent-ID of the device-agent realization.

```
package a3me;

import a3me.asn.a3mecontent.DaID;
import a3me.info.NeighborsInfo;

/**
 * IDeviceAgent defines local API for a DeviceAgent
 *
 * @author aherzog
 */
public interface IDeviceAgent{

  public DaID getDaID();

  public AbstractMessageHandler getMessageHandler();

  public ComComponent getComComponent();

  /**
   * ClassificationStorage was replaced by InfoStorage but this lead to an Java-VM Error on SunSpots.
   * So Sunspots now use the old ClassificationStorage, while workstation and android start using
       InfoStorage.
   */
  public IStorage getInfoStorage();

  public IServiceDirectory getServiceDirectory();

  public NeighborsInfo getNeighborsInfo();
}
```

Listing 23: Device-agent Interface

5.1.1.2 IComm Interface

The IComm interface (Listing 24) defines the send and listen methods for a communication interface, a method to get the name of the communication interface and a method to get the address of the current device used for this communication interface.

```
/*
* IComm.java
*
* Created on 15.08.2008, 17:23:00
```

```
*
*/

package a3me;

import a3me.asn.a3mecontent.Address;
import a3me.asn.a3mememessage.Message;
import java.io.IOException;

/**
 * Interface to be implemented for different plattforms/hardware/comm.technology.
 *
 * @author aherzog
 */
public interface IComm {

    public boolean send(Message msg);

    public void listen(IMessageListener ml) throws IOException;

    public String getName();

    public Address getAddress();
}
```

Listing 24: Communication Component Interface

5.1.1.3 IMessageHandler Interface

The IMessageHandler interface (Listing 25) defines all the typical methods needed to deal with a message.

```
/*
 * A3ME framework
 * author: Arthur Herzog
 */

package a3me;

import a3me.asn.a3mecontent.DaID;
import a3me.asn.a3mecontent.Resultset;
import a3me.asn.a3mememessage.ConversationID;
import a3me.asn.a3mememessage.Message;

/**
 *
 * @author aherzog
 */
public interface IMessageHandler {

  public void messageReceived(Message amsg);

  public boolean send(Message amsg);

  /**
   * Is used to send advertisement info messages and for answering queries.
   *
   * @param replyTo DaID to which the answer shall be sent.
   * @param oConvID ConversationID for the answer message.
   * @param rs      Resultset for the query.
   * @param iSeqNr  Sequence Nr for continuous queries.
   */
  public void sendInfo(DaID daidTo, ConversationID oConvID, Resultset rs, int iSeqNr);
}
```

Listing 25: Message Handler Interface

5.1.1.4 IMessageListener interface

The IMessageListener interface (Listing 26) offers a method to signal when a message is received to which the implementing class can react.

```java
/*
* IMessageListener.java
*
* Created on 15.08.2008, 14:48:24
*
*/

package a3me;

import a3me.asn.a3memessage.Message;

/**
 * IMessageListener interface.<p>
 *
 * An interface for listening to A3ME messages.
 *
 * @author Arthur Herzog
 */
public interface IMessageListener {

    /**
     * This method is called to signal message reception.
     *
     * @param m the received message
     */
    public void messageReceived(Message m);

}
```

Listing 26: Message Listener Interface

5.1.1.5 IQueryHandler interface

The IQueryHandler interface (Listing 27) defines basic methods to deal with a query.

```java
/*
 * A3ME framework
 * author: Arthur Herzog
 */
package a3me;

import a3me.asn.a3mecontent.Resultset;
import a3me.asn.a3memessage.ConversationID;
import a3me.asn.a3memessage.Message;
import a3me.asn.a3meontology.A3ME_code;
import java.util.Enumeration;

/**
 * @author Arthur Herzog
 */
public interface IQueryHandler {

  public void addQuery(Message amsg);

  public void answerQuery(Message msg, boolean includeDataDescriptor, int iSeqNr);

  public Resultset getClassification(A3ME_code code);

  public Enumeration getQueryIDs();

  public Message getQueryMsg(ConversationID oConvID);

  public void stopQuery(ConversationID oConvID);
}
```

Listing 27: Query Handler Interface

addQuery(Message amsg): defines a new query which is contained in the message.

answerQuery(Message msg, boolean includeDataDescriptor , int iSeqNr): generates the answer message and triggers its transmission via the Com-Component.

getClassification (A3ME_code code): Returns a resultset containing the classification entries for given A3ME code.
getQueryIDs(): Returns list of QueryIDs.
getQueryMsg(ConversationID oConvID): Returns the original message from which the query was created.
stopQuery(ConversationID oConvID): Stops the specified query.

5.1.1.6 IService interface

The IService interface (Listing 28) defines basic methods for a service implementation.

```
/* A3ME — Device–Agent based Middleware for Mixed Mode Environments */
package a3me;

import a3me.asn.a3mecontent.Data_value;
import a3me.asn.a3mecontent.Record;
import a3me.asn.a3meontology.A3ME_code;
import a3me.info.InfoEntry;

/**
 * Service interface defining required methods to be implemented by all A3ME services.
 * @author aherzog
 */
public interface IService {

  /** Starts the service. */
  public void start();

  /** Stops the service. */
  public void stop();

  /**
   * Returns the A3ME code of the capability to which the service is related.
   * @return A3ME code or -1 if the service is not related to a capability of the device.
   */
  public A3ME_code getRelatedCapability();

  /**
   * Returns the InfoEntry for the service.
   * @return InfoEntry
   */
  public InfoEntry getInfoEntry();

  /**
   * Returns the current data value if applicable.
   * For example in the case of an sensor related service it would be the actual sensor reading.
   * @return Data_value or null if not applicable.
   */
  public Data_value getData();

  /**
   * Returns the machine readable description of the service.
   * @return Data_value containing the m2m description.
   */
  public Data_value getM2MDescription();

  /**
   * Executes the given command with provided parameters on the current service.
   * @param command - command to execute.
   * @param params - optional set of parameters.
   */
  public void execute(int command, Record params);
}
```

Listing 28: Service Interface

5.1.1.7 IServiceDirectory interface

The IServiceDirectory interface (Listing 29) defines basic methods for a service directory component of an A3ME realization.

```
/*
 * A3ME framework
```

```
 * author: Arthur Herzog
 */

package a3me;

import a3me.asn.a3meontology.A3ME_code;
import java.util.Vector;

/**
 *
 * @author aherzog
 */
public interface IServiceDirectory {

  /**
   * Adds a local service.
   *
   * @param se Object which implements IService interface.
   * @return local id for the service.
   */
  public int addService(IService se);

  /**
   * Deltes a service with given local id.
   *
   * @param iID local ID
   */
  public void deleteService(int iID);

  /**
   * Returns Object with local id iID, which implements IService interface.
   *
   * @param iID
   * @return
   */
  public IService getService(int iID);

  /**
   * Returns a Vector of IService Objects with services for given capability (or subcapability).
   *
   * @param code capability which the service shall be ralated to.
   * @return Vector of IServices
   */
  public Vector getServicesForCapability(A3ME_code code);

  /**
   * Returns a Vector of IService Objects which belong to the given A3ME code.
   *
   * @param A3ME code
   * @return Vercotr of IService Objects
   */
  public Vector getServices(A3ME_code code);

}
```

Listing 29: Service Directory Interface

5.1.1.8 IStorage interface

The IStorage interface (Listing 30) defines basic methods to interact with the information storage component of an A3ME realization.

```
/* IStorage.java */
package a3me;

import a3me.asn.a3meontology.A3ME_code;
import a3me.info.InfoEntry;
import a3me.ontology.ClassificationImpl;
import java.util.Vector;
```

```
/**
 * Information Storage interface defines methods to interact with the
 * information storage of an device-agent.
 *
 * @author aherzog
 */
public interface IStorage {

  /**
   * returns the used ClassificationImpl.
   */
  public ClassificationImpl getClassification();

  /**
   * Adds info for a concrete device, capability, etc.
   *
   * @param code
   * @return local ID for the classificationEntry, -1 if code is not valid.
   */
  public int add(A3ME_code code);

  /**
   * Adds info for a concrete device, capability, etc.
   *
   * @param code
   * @param sName
   * @return local ID for the classificationEntry, -1 if code is not valid.
   */
  public int add(A3ME_code code, String sName);

  /**
   * @param code
   * @param sName
   * @param sDescription
   * @return
   */
  public int add(A3ME_code code, String sName, String sDescription);

  /**
   * @param infoEntry
   * @return
   */
  public int add(InfoEntry infoEntry);

  /**
   * @param code
   * @param sName
   * @return
   */
  public int getID(A3ME_code code, String sName);

  /**
   * deletes info entry.
   *
   * @param id device-local id of the classification entry
   */
  public void delete(int id);

  /**
   * deletes info entry.
   *
   * @param code
   * @param sName
   */
  public void delete(A3ME_code code, String sName);

  /**
   * Returns Vector of Short values of contained classification codes
   * which are element of code.
   *
```

```
    * @param code
    * @return
    */
   public Vector getCodesFor(A3ME_code code);


   /**
    * Returns first ID which is element of code.
    * Which one from multiple matching entries is not definite.
    *
    * @param code
    * @return
    */
   public int getID(A3ME_code code);

   /**
    * Returns Vector of Integer values for classification ids
    * which are element of code.
    *
    * @param code
    * @return
    */
   public Vector getIDsFor(A3ME_code code);

   /**
    * @param id
    * @return
    */
   public A3ME_code getCode(int id);

   /**
    * @param id
    * @return
    */
   public String getName(int id);

   /**
    * @param id
    * @return
    */
   public String getDescription(int id);

   /**
    * @param id
    * @return
    */
   public InfoEntry getInfoEntry(int id);
}
```

Listing 30: Information Storage Interface

5.1.2 Common Components Implementation

These are common classes which implement the functionality shared across different A3ME realizations. They are implemented using Java 1.4 version and can therefore be used for all the involved Java based implementations.

The parts which are common across the used Java version and implementations of A3ME. Other device dependent parts of the implementation are defined as abstract and will be implemented in the subclasses of the corresponding abstract class. Classes containing abstract methods are defined as abstract and have to be extended for the Java version and device specific implementations.

The internal structures and data are also defined here and reused in the individual device specific subclasses.

5.1.2.1 AbstractMessageHandler Implementation

AbstractMessageHandler implements methods, which are independent of the device and Java version used.

boolean acceptsMessage(Message m): Accepts all messages, therefore always returns true.

void answerRequest(Message msg, Resultset rs, int iSeqNr): Callback function to be called by QueryHandler when the answer is ready to be send.

protected abstract NeighborsInfo getNeighborsInfo(): Returns a reference to the NeighborsInfo.
long getNextCoversationNr(): Generates next conversation number for this DA.
protected static boolean isContainedIn(Address adr1, Enumeration eAddresses) : Checks whether adr1 is contained in eAddresses.
protected static boolean isContainedIn(DaID daid, Enumeration eDestAddresses) : Checks whether one of the daid's addresses is contained in the eDestAddresses.
protected boolean preHandling(MessageContainer msgCont) : Executes the following methods before handling the message:

- check: is msg == null.
- check: isMyOwnMessage.
- check: isDuplicateOrOld.
- forwardIfNeeded.
- check: isForMe.

Should be executed inside handle() method before handling the message, handling should not be continued if false is returned.
boolean send(Message msg) : Send the message through all available communication interfaces.
void sendInfo(DaID daidTo, ConversationID oConvID, Resultset rs, int iSeqNr) : Is used to send advertisement info messages and for answering queries.
protected abstract void handle(): In this method the actual handling of the message shell be executed.
abstract void messageReceived(Message amsg) : This method is called to signal message reception.

5.1.2.2 QueryHandler Implementation

The QueryHandler class implements the IQueryhandler interface. For each incoming query the following steps are executed:

1. Get what is requested.
2. apply conditions from the *WHERE* part of the query.
3. give result to message handler (using *answerQuery(...)* method).
4. if periodic, repeat query after given period (in *addQuery(...)* method).

The query can be a data-query or a service call. The operations to be executed differ for these two types in the steps 1, 2 and 3.

Data-query:

1. In the case of a data query the request contains a list of data-descriptors. Each data-descriptor is composed of an query-object – classification entry from the A3ME classification – and an info-type (See section 4.11.9).
   For the result-set composition the following steps are executed:
   a) Identify the contained query-objects. Each query-object is treated as a relation and contains columns, which correspond to the info-types requested or used (e.g. in the *WHERE* part) in the query for the given query-object.
   b) Apply conditions from the *WHERE* part.
   c) Identify query-objects which are subsets of another query-object present in the query and merge the two relations as a natural join.
   d) Execute a cross join on all the remaining relations.
2. The conditions from the *WHERE* part of the query have for performance issues already been applied in step 1, to reduce the size of relations as soon as possible before executing the expensive merge operations.
3. The computed result-set is returned to the message handler, which forwards the result to the requester.
4. The query is scheduled for repetition after given period, if *PERIOD* is specified in the request.

Service-call:

1. In the case of a service call we have to identify the requested services. Since the service can be specified implicitly by the related capability, there can be more than one service.
2. Apply conditions from the *WHERE* part of the query if applicable and remove services from the applicable services set if conditions are not satisfied.
3. a) Execute the requested command with given parameters on all the services remaining in the set of applicable services.
   b) Notify the message handler whether an service command was triggered. The execution of the service is not waited for to end.
4. Repeat query after given period, if REPETITION part is specified in the request (in *addQuery(...)* method).

5.1.2.3 AbstractService Implementation

The abstract class AbstractService shall serve as a base class for all services. It is initialized with the parameters

- A3ME_code oRelatedCapabilty which points out the capability to which the service is related, if it is the case and
- InfoEntry oInfoEntry containing the code, name, description and m2m description of the service.

The following three methods are common for all the implementers of IService and are therefore implemented in this abstract class.

InfoEntry getInfoEntry() : Returns the information about this service.

public String getM2MDescription() : Returns the machine readable description of the service if available. Services which don't have m2m description return null.

A3ME_code getRelatedCapability() : Returns the capability to which this service is related or null if it is not the case.

The methods start, stop, execute are specific to the services and have to be implemented in the subclasses.

5.1.2.4 ASNCoder Implementation

The ASNCoder class is responsible for encoding and decoding of ASN.1 messages. It is statically initialized with the unaligned Packed Encoding Rule (PER) encoder and offers the encode and the decode methods.

The ASNCoder uses the Java version specific ASN.1 encoding library from OSS Nokalva, Inc.[40].

5.1.2.5 ComComponent Implementation

Meta communication component for a Device-Agent: contains the real communication components. Should not be used directly but through MessageHandler classes.

Offers the methods *addCommInterface* and *removeInterface* with an instance of *IComm* interface as parameter to add or remove the corresponding communication interface.

5.1.2.6 QueryTask Implementation

The QueryTask is an TimerTask which handles an Query in a separate Task if needed in a repetitive way. It is initialized with the query message containing the query and the reference to the query handler to report results.

5.1.2.7 QueryDeleteTask Implementation

The QueryDeleteTask contains the code to stop an query specified as parameter. It is then used in the java.util.Timer as parameter to be executed after specified time and therefore stop the query. This is used for queries where in the *REPETITION* part of the query the *DURATION* also has been specified.

5.1.2.8 Utilities Implementation

Implements often used functionalities for conversions and for String representations of data.

5.1.3 Special Problems: Java Libraries Conflicts

On the workstation we needed to implement javax.microedition.io.Connector for the Bluetooth and for the Sun SPOT communication component. For both communication components we had to include a Java library. The problem occurred since both libraries had a class named javax.microedition.io.Connector. Depending on which library appeared first in the Java class path, the other communication component complained about the wrong javax.microedition.io.Connector implementation.

The problem was solved by addressing the concrete implementing class of the javax.microedition.io.Connector interface in each library. Since those were named differently, Java VM could locate and use those now.

5.2 A3ME for Sun SPOTs

For the first A3ME implementation the Sun SPOT [24] platform[41] (Figure 26) was selected. Project Sun SPOT (Sun Small Programmable Object Technology) is a snapshot of ongoing research in Oracle Labs (formerly Sun Labs). This Platform runs a version of JavaME [42], called Squawk, that supports Connected Limited Device Configuration (CLDC) 1.1 and Mobile Information Device Profile (MIDP) 1.0.

An advantage of this platform is the easy debugging possibility through usage of system printouts, when a Sun SPOT is connected via USB to the computer.

[40] OSS Nokalva, Inc. company web page http://www.oss.com/

[41] Sun SPOT project web page: http://www.sunspotworld.com.

[42] Java ME web page: http://www.oracle.com/technetwork/java/javame/.

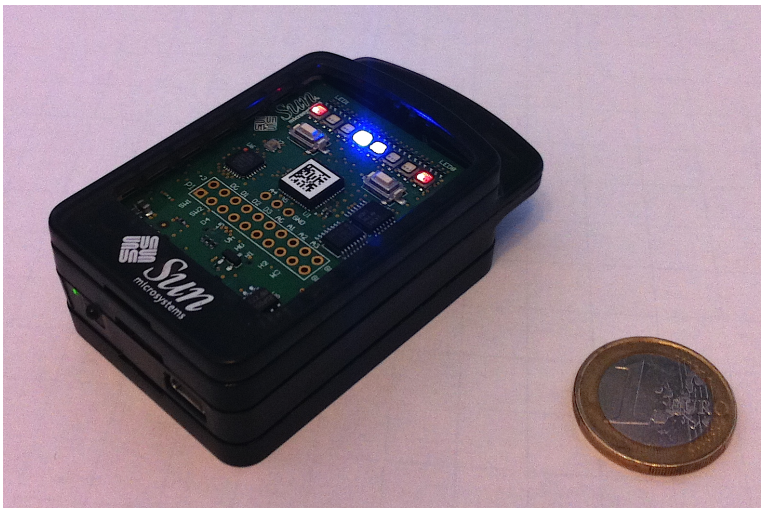


Figure 26: Sun SPOT Sensor Node

5.2.1 Sun Spot Platform Overview

Sun SPOT device (revision 6) [22] is a sensor node of the size 4,2 cm width x 7,1 cm length x 2,3 cm depth. The main board is equipped with a

- 32 bit 180 MHz ARM9 processor,
- 4 MB Flash memory,
- 512 kB RAM memory,
- USB Mini Type B connector,
- 8-bit Atmel Atmega88 microcontroller for power management,
- 64-bit ms real time clock (on microcontroller),
- CC2420 2.4GHz IEEE 802.15.4 compliant radio transceiver,
- battery voltage sensor,
- temperature sensor.

It is shipped in two variants base station SPOTs and eSPOTs. The base station SPOT has no battery, sensors and leds and is used as gateway for a PC to communicate with other Sun SPOTs wirelessly. The eSPOTs are equipped with [22]:

- a rechargeable LI-ION battery with 3.7V 720mAh,
- 8 three-color leds,
- temperature sensor,
- light sensor,
- three-axis acceleration sensor,
- 2 button switches.

Sun SPOTs run Squawk VM which is an implementation of Java ME. The Sun SPOT applications are programmed in Java as MIDlets and can then be deployed.

5.2.2 Sun SPOT Communication

Sun SPOTs communicate via the CC2420 [16] radio transceiver in the 2.4 GHz ISM band. It uses the 802.15.4 MAC protocol. The usable frequency spectrum is divided into 16 channels. The channel to be used for communication can be changed at runtime. In our implementation we use the default channel for all Sun SPOTs and do not have to run device discoveries to find out the channels used by the SPOTs.

5.2.3 Device-agent Realization

The Device-agent realization for the Sun SPOT platform (SunSpot-DA) is implemented as a MIDlet. A MIDlet is an application that uses the Mobile Information Device Profile (MIDP) of the Connected Limited Device Configuration (CLDC) for the Java ME environment. It provides a well-defined lifecycle controlled via methods of the MIDlet class.

The SunSpot-DA is a basic level device-realization: it implements all the capabilities and additionally the services related to the hardware capabilities of this platform. The sensor readings can be queried and the LEDs can be controlled via service calls.

The implementation reuses most parts from the common A3ME implementation described in section 5.1.2. Only the Info-Storage component from the common parts was incompatible with this platform because of class inheritance issues and had to be implemented in a simpler way.

5.2.4 GUI

The Sun SPOT platform does not have an output screen so there is no graphical user interface as such. The only available visualization on the a Sun SPOT device itself is through the eight three-color leds.

For debugging and logging it is also possible to print messages to standard out which can be displayed on a workstation, which is either connected directly via USB cable or indirectly through a Sun SPOT base station.

5.3 A3ME for a Workstation

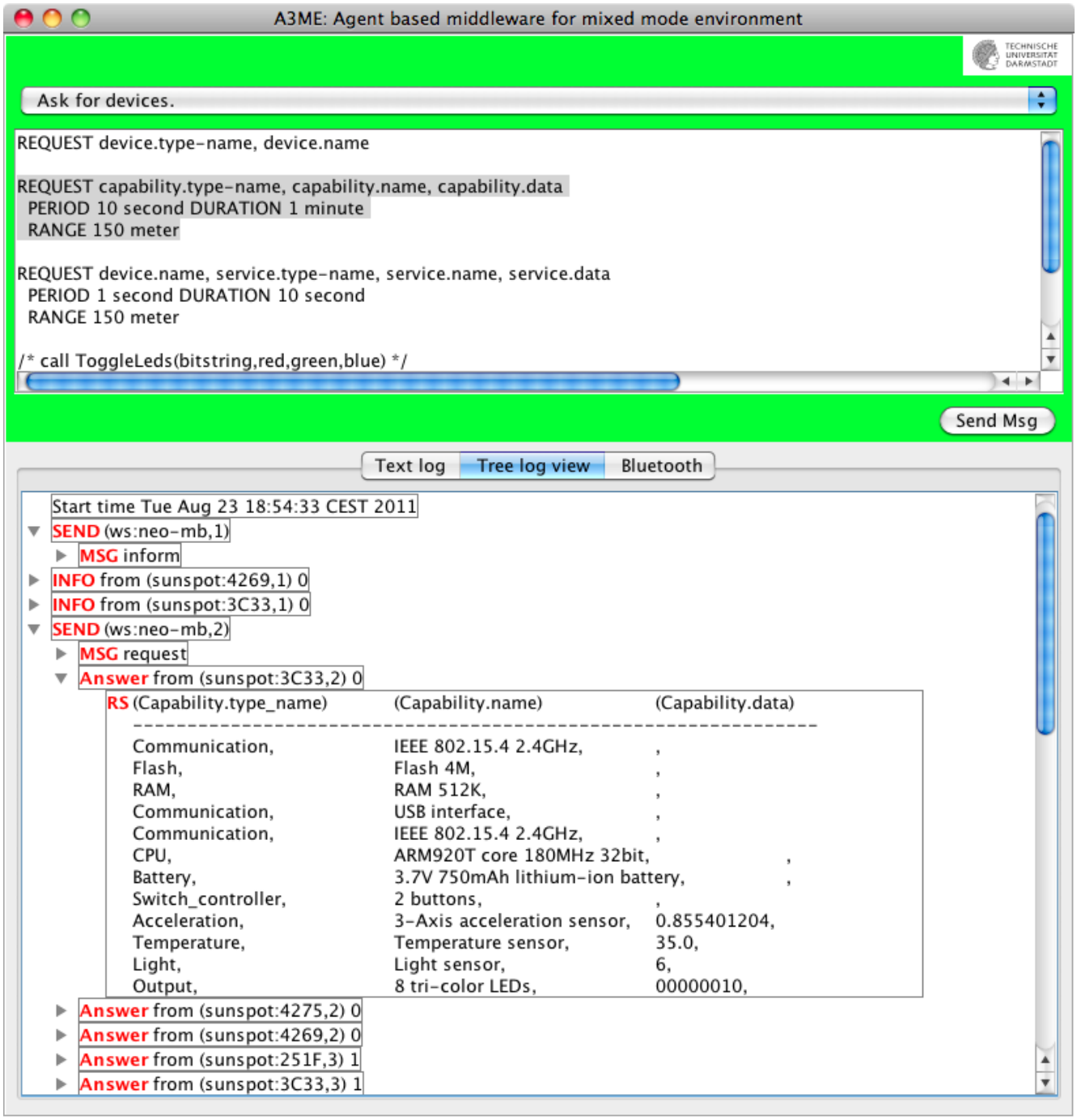


Figure 27: A3ME GUI on a Workstation

In parallel to the implementation of the SunSpot-DA we implemented the DA for the workstation to be able to test interactions between these two platforms. This included the communication component to communicate with Sun SPOTs via a connected Sun SPOT base station. Later we added further communication components to communicate via Bluetooth with the android smartphone and with the tmote-sky platform running Contiki.

First of all a simple user interface on a workstation was implemented to interact with other type of devices. Later this simple interface was further extended to a graphical user interface for easier monitoring of the interactions with other devices in the network.

5.3.1 Device-agent Realization

The Device-agent Realization for a workstation (WS-DA) is implemented in Java Standard Edition 6. Here we extend the core implementation described in section 5.1.2. It implements the descriptive capabilities of the workstation. Since the workstation usually does not have sensor and actuator capabilities it does not offer sensor or actuator related services. Its main purpose here is to deal as a user interface and as a bridge between different communication technologies available on the workstation.

5.3.2 GUI

The graphical user interface (GUI) for the workstation Device-agent (Figure 27) is divided into input area (green background) and output area (gray background).

5.3.2.1 User Inputs

The input area allows the user to enter commands. For this the user has two options:

- **pull-down list** with predefined commands:
  - ask for devices,
  - ask for capabilities,
  - ask for services.
- **text input field**.

The predefined commands send a query asking for devices, capabilities or services from all devices in range through all available communication interfaces. In the input field the user can enter queries in A3ME-QL. To simplify the definition of the queries the input field already contains typical queries, which can be edited and extended.

5.3.2.2 Outputs

The output part of the GUI offers three tabs:

- Textual log,
- Tree based representation of the exchanged messages (Figure 27),
- Textual log of the Bluetooth communication interface.

The most user friendly is the tree-based representation of the exchanged messages. Here we can see the send queries tagged with the query ID. Attached to it is the ASN.1 definition of the message. The answers to each query are added as sub branches to the corresponding query tier (see figure 27). Results of data-queries are represented as tables.

5.3.3 Sun SPOT Communication Interface

To enable the workstation to communicate with Sun SPOT sensor nodes (section 5.2.1) the workstation is connected to a Sun SPOT base station, which is usually a Sun SPOT without sensor board and battery. The purpose of the Sun SPOT base station is to allow applications running on the Host to interact wirelessly with other Sun SPOTs [24]. The physical arrangement is shown in Figure 28. The base station communicates with other Sun SPOTs via the 802.15.4 [15] communication standard.

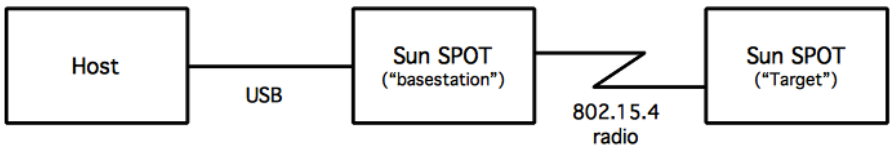


Figure 28: Workstation Communicates with Sun SPOTs through a Connected Base Station [24].

The Sun SPOT base station is running in shared mode allowing other applications besides workstation-DA to use it and to communicate with Sun SPOTS independent of the A3ME framework.

5.3.4 Bluetooth Communication Interface

Most workstations nowadays have a Bluetooth interface. Therefore it is quite reasonable to implement this communication capability for the device-agent representing a workstation.

The devices-agent realization for the workstation is written in Java, so it usually does not matter which operating system the workstation is running. In the case of the Bluetooth interface it matters what operation system the workstation is running. To use the Bluetooth interface the device-agent implementation requires a Java library which allows to access the corresponding hardware.

Most available libraries are written only for one specific Operation System (OS). This would mean that we have to use different libraries for different OSs. There are only few supporting multiple OSs. For the prototype implementation we decided to use BlueCove library. BlueCove[43] is a JSR-82 J2SE implementation that currently interfaces with the Mac OS X, WIDCOMM, BlueSoleil and Microsoft Bluetooth stack found in Windows XP SP2 and newer. Originally developed by Intel Research and currently maintained as open source by volunteers.

The disadvantage of BlueCove is the one common for many open-source projects that it is not maintained regularly. For our implementation there is an issue of reopening a stream once it was closed. This leads to an unstable communication via Bluetooth. But for the prototypical implementation it is enough to show the functionality of the concept as such.

5.3.5 UPNP Communication Interface

While the communication interfaces for Bluetooth and for SunSpot are components which realize the corresponding communication hardware binding and the protocols, the communication interface for Universal Plug and Play (UPNP) (3.5.5) is different. The UPNP Communication Interface realizes a connection to another framework – UPNP.

This is realized by:

- Register and announce the device agent as UPNP device, so it can be discovered by other UPNP devices.
- Match the A3ME representation of devices and services to UPNP types.
- Register the properties and services of the device in its UPNP representation.
- Translate and Forwarding A3ME request to the UPNP.
- Translate the incoming UPNP answers and notifications into A3ME descriptions and forward those to the message handler of the device-agent.

For the implementation of the UPNP Communication Interface (UPNP com-interface) we use the **Cling**[44] - Java UPnP library. The UPNP com-interface starts a UPNP server which announces the represented device as *a3me-da-workstation*. We set the UPNP device type to "A3ME, 1" and set the manufacturer details property to "Arthur Herzog". This enables the UPNP com-interface to answers UPNP requests received from other UPNP devices in the network.

When ever the *send* method of the UPNP com-interface is called we check if the message to send contains a data or service request and translate it then into an UPNP request. The UPNP request results are then filtered, translated into the A3ME representation and reported then to the Message-Handler component as received answer messages.

For the translation of the received UPNP information to A3ME data structures we used the matching shown in table 10.

UPNP	A3ME Infotype
Device DisplayString	device.name
Device Type + ManufacturerDetails	device.description
Device Udn	device.id_nr
Device FriendlyName	device.data
Device DescriptorURL	device.m2m-description
ServiceId	service.name
ServiceType	service.description
Service DescriptorURL	service.m2m-description

Table 10: Information Matching between UPNP and A3ME

Figure 29 shows the results of two A3ME queries asking for devices and services received from UPNP devices via the UPNP com-interface.

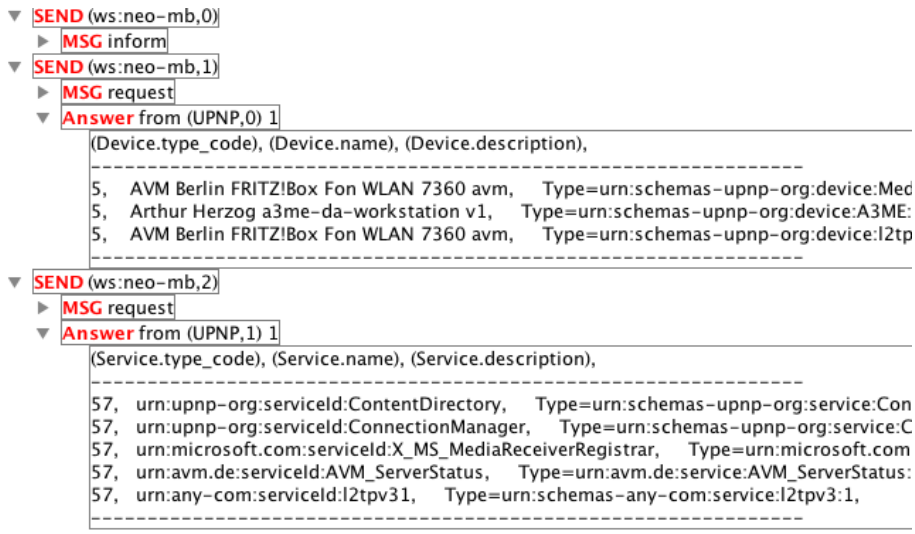


Figure 29: Workstation GUI Output with Results for Devices and Services Requests from UPNP Devices.

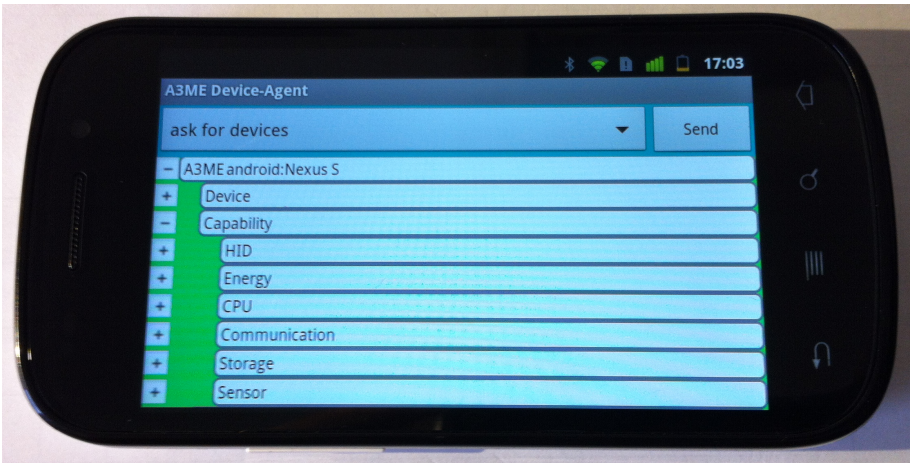


Figure 30: A3ME Running on Android OS based Nexus S Smartphone.

5.4 A3ME App for Android Platform

Smartphones became the ultimate human interface device (HID) today. Therefore we decided to implement a prototypical A3ME device-agent realization for a smartphone and to use it as a HID. Even though nowadays smartphones have a variety of sensors and communication capabilities on board we consider their major functionality to be the human interface. For the prototypical implementation we decided to use Android platform, since it supports a growing amount of smartphone and other devices and has a very open policy considering different APIs.

The Android platform offers very convenient development environment[45]. It runs Dalvik virtual machine which provides most of the functionality available in the core libraries of the Java programming language. This means that the implementation is done in the Java programming language. As platform we use a Samsung Nexus S smartphone (figure 30).

Since the programming language for Android Platform is Java, we can reuse the core Java 1.4 implementation described in section 5.1.

5.4.1 Smartphone's Hardware Overview

For the evaluation we use the Samsung Nexus S smartphone [141]. It has a 1 GHz ARM Cortex A8 based CPU, 512 MB of RAM and 16 GB Flash memory. It has a 10 cm touchscreen with a resolution of 800 x 480 pixels making it a reasonable user interface device to interact with other devices.

[43] BlueCove - Java library for Bluetooth web site: http://code.google.com/p/bluecove/.

[44] Cling - Java/Android UPnP library and tools web site: http://4thline.org/projects/cling/

[45] Android Developers web page: http://developer.android.com.

Samsung Nexus S smartphone has the following communication capabilities:

- UMTS, GPRS,
- Bluetooth 2.1+EDR,
- WLAN,
- NFC,
- USB.

It also has the following build-in sensors:

- GPS positioning sensor,
- Foto/video camera,
- Compas,
- Rotation sensor,
- Microphone,
- Light sensor.

The smartphone is running Android operating system version 4.0.4.

5.4.2 Device-Agent Realization

Based on Java core implementation we implement the device-agent realization for Android platform (Android-DA) as an service in the Android Operating System. This allows the device-agent to run in the background without a GUI.

On initialization the Android-DA retrieves the available capabilities of the current device and registers them as A3ME capabilities. While the information about available sensors can easily be queried, other capabilities like CPU, keyboard, display etc. all have to be collected separately at different locations. The information about camera, which is classified in A3ME as a sensor, has to be collected at a different location.

5.4.3 GUI

The graphical user interface (GUI) is implemented as an Android activity and is connected to the device-agent service. Here we implemented two views log-view and an info-view. User can switch between the two views by swiping to left or right.

The log-view displays text messages for all events and actions inside the Android-DA:

- command sent,
- messages received,
- errors occurred,
- debugging messages.

The info-view is the main interface for the user. It shows the user available information (Figure 39) and offers the possibility to send out requests to discover other devices and their capabilities. In section 6.2.6 we demonstrate the usage of this application.

After initialization the info-view displays only the information about the device itself. The user has the possibility to trigger one of the predefined commands:

1. ask for devices,
2. ask for capabilities,
3. ask for services,
4. call service blink LEDs,
5. call service LEDs off.

First three commands send out queries for information about devices, their capabilities and services offered. The commands 4 and 5 send out service calls related to output capabilities and let the devices which offer matching services to blink or to turn off their led.

5.4.4 Bluetooth Communication Interface

In Android-DA we implemented the Bluetooth communication component to allow direct communication with other device-agents capable to communicate via Bluetooth, e.g. the Workstation-DA or other Android-DAs.

Bluetooth Communication Interface is implemented as an Android service and allows the device-agent to connect to it. It offers an A3ME Bluetooth service which waits for incoming Bluetooth connections. On DA startup and on user request it searches for other Bluetooth devices in range (figure 31). Once it finds another Bluetooth device it searches for A3ME services on it.

5.5 A3ME Module for Robot Operating System

We implemented a Robot Operating System (ROS) (see section 3.3.1) module which realizes the A3ME Core DAI. The module is implemented in Java using Java Native Interface (JNI). It allows other ROS modules to interact with devices

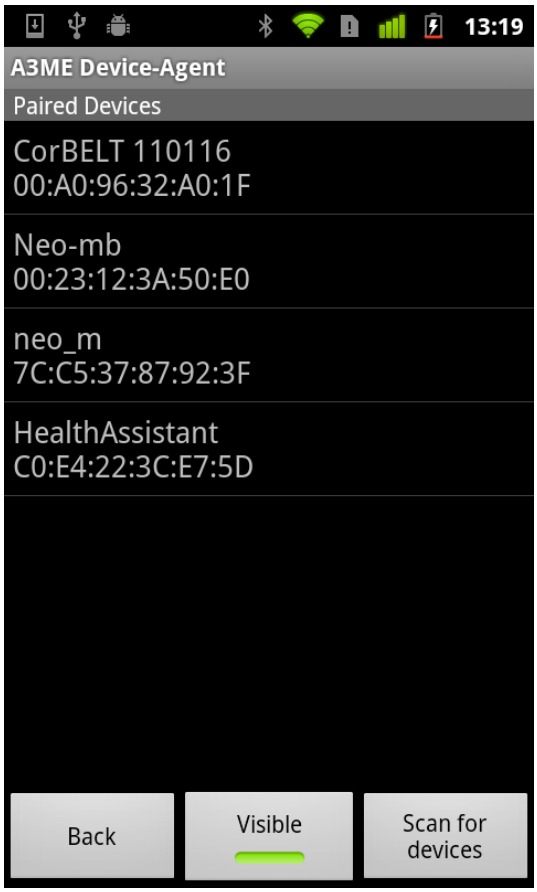


Figure 31: Bluetooth Paired Devices on Android Device-Agent

through the A3ME framework. So it allows to use information offered through the A3ME DAI, which includes the information about the device itself and about other known devices through the A3ME framework. Other ROS modules can trigger predefined queries or define their own ones to be send. The results can be listened to through the 'a3me' pub-sub topic channel and can be additionally forwarded directly via ROS inter module messages.

5.6 A3ME for TelosB Sensor Platform

To proof the low hardware requirements and the functionality of A3ME we also implement the framework for the TelosB platform using the Contiki Operating System[46].

5.6.1 TelosB Platform Overview

The TelosB platform (Figure 32) is a sensor node platform produced by multiple manufacturers (Crossbow, MoteIV et al.) and is widely used in the research community. TelosB platform from Crossbow (now distributed by Memsic Inc.) has the following properties [14] [123]:

- IEEE 802.15.4 compliant radio chip,
- 250 kbps, high data rate radio,
- 8MHz Texas Instruments MSP430 F1611 16-bit microcontroller with 10 kB RAM and 48 kB Flash,
- Integrated onboard antenna,
- Data collection and programming via USB interface,
- Open-source operating system,
  - Tiny OS or
  - Contiki OS,
- integrated sensors:
  - temperature,
  - visible light,

[46] Contiki Operating System web page: http://www.contiki-os.org.

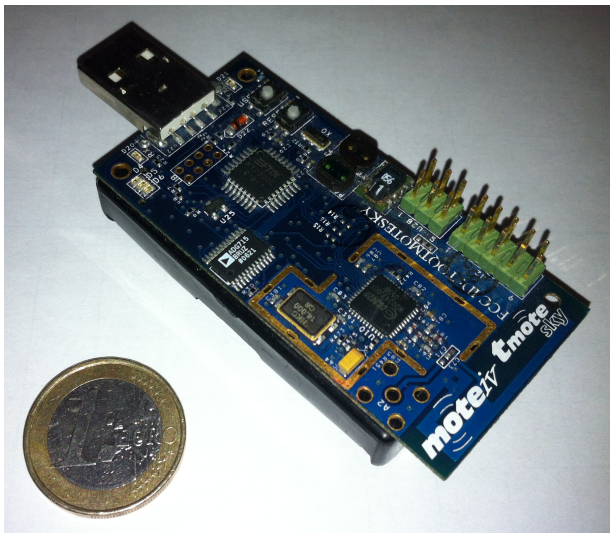


Figure 32: TelosB Sensor Node from MoteIV Company

  - visible and infrared light,
  - humidity,
  - battery voltage,
- power supply via:
  - USB or
  - two AA batteries.

5.6.2 TelosB Communication

The TelosB sensor nodes communicate via the CC2420 [16] radio transceiver in the 2.4 GHz in the ISM band. IEEE 802.15.4 MAC protocol specifies 16 channels. For our implementation in Contiki OS we specify the same channel for all nodes at compile time. Since all nodes use the same channel we don't have to discover which channel is used. The used B-MAC protocol [132] allows to send and receive messages without previous neighbor discovery. To achieve this the protocol uses carrier sense media access capability and an adaptive preamble sampling scheme.

5.6.3 Device-Agent Realization

We have implemented basic device-agent functions for the TelosB platform. The ASN.1 encoding and decoding is done with the library provided by the OSS Nokalva Inc. Alessandro Triglia from the OSS Nokalva company ported and striped down their C ASN.1 library to reduce the size of the library. To reduce the library size some limitations[47] were added to the ASN.1 definition:

- Default values are not supported (DEFAULT is treated as OPTIONAL).
- The REAL type is not supported.
- Size of the encoding control information for the ASN.1 specification: 8 KB max (this is sufficient for up to about 8000 lines of ASN.1 code).
- Size of the encoded message: 64 KB max.
- Size of the decoded message: 64 KB max.
- Size of each SEQUENCE OF, OCTET STRING, BIT STRING, or IA5String: 65535 elements max (for a BIT STRING, this means 8 KB max).
- All SEQUENCE OF types must have a size constraint (for example, "SEQUENCE (SIZE (0..10000)) OF INTEGER" is allowed, whereas "SEQUENCE OF INTEGER" is not allowed).
- All OCTET STRING, BIT STRING, and IA5String types must have a size constraint (for example, "OCTET STRING (SIZE (0..10000))" is allowed, whereas "OCTET STRING" is not allowed).
- The contents constraint ("OCTET STRING (CONTAINING (....))") is not supported.
- The only supported character string type is IA5String (7 bits per character); note that UTF-8 strings can be specified as "OCTET STRING (SIZE (.....))", where the size constraint refers to the number of bytes, not to the number of Unicode characters.
- The permitted-alphabet constraint ("IA5String (FROM (....))") is not supported.

[47] Source: personal email from Alessandro Triglia from OSS Nokalva, Inc. on September 11. 2011.

- Range of INTEGER types: unsigned integers 0..4294967295, signed integers -2147483648..2147483647.
- Extension markers may be present in type definitions but the type definitions must not contain any extension additions (if the decoder finds a value of an extensible type with the extension bit set, it will return an error).
- GeneralizedTime data type can not be used.

After applying this limitation we managed to implement the encoding and decoding A3ME messages and to exchange them among the nodes. By attaching a TelosB sensor node as gateway to a workstation we can enable a workstation device-agent to send and receive messages from the sensor nodes. This also enables a Workstation-device-agent to relay messages between its other communication interfaces and the 802.15.4 communication interface used by TelosB nodes.

5.7 A3ME for Z1 Sensor Platform

Z1 is a sensor node platform (Figure 33) produced by Zolertia company[48]. Z1 device is very similar to the TelosB platform (section 5.6.1).

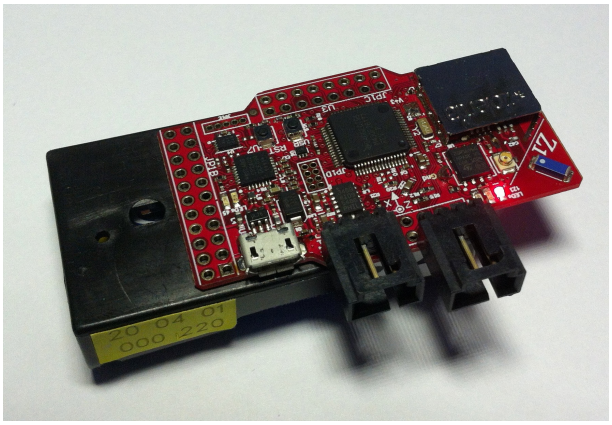


Figure 33: Z1 Sensor Node from Zolertia Company

5.7.1 Z1 Platform Overview

Zolertia Z1 sensor node has a newer MSP430 processor and an additional acceleration sensor. The remaining configuration is similar as on the TelosB platform. [25]

- IEEE 802.15.4 compliant radio chip,
- 250 kbps, high data rate radio,
- 16MHz Texas Instruments MSP430 F1611 16-bit microcontroller with 10 kB RAM and 48 kB Flash,
- Integrated onboard antenna,
- Data collection and programming via USB interface,
- Open-source operating system,
  - Tiny OS or
  - Contiki OS,
- integrated sensors:
  - temperature,
  - visible light,
  - visible and infrared light,
  - humidity,
  - battery voltage,
- power supply via:
  - USB or
  - two AA batteries.

5.7.2 Device-Agent Realization

For the realization of the A3ME for this platform we reused the Contiki OS implementation developed for TelosB platform.

[48] Zolertia web page: http://zolertia.com

6 Evaluation

The A3ME framework is designed to work with different types of devices and to be used for very different tasks. In the following section we describe experiments which demonstrate some aspects and use cases for A3ME. First, in section 6.1 we demonstrate how expressive and self-descriptive the messages used in the A3ME framework are. These messages can be constructed dynamically, verified against the ASN.1 definition and encoded into compact byte streams for transmission. This verifies one part of our goal of building a generic solution (1.2.3) by allowing our messages to transport very different types and combinations of structured information and it also verifies the requirement R5 "*Low Hardware Requirements.*" (2.5) by reducing the amount of data needed to be transmitted.

Second, in section 6.2 we present a series of experiments to demonstrate the realization of different in section 2 defined requirements by the A3ME framework and to show possible application scenarios for the use of the framework. Here we run different experiments with varying sets of participating devices and device types and present measurement results, which were enabled by the use of A3ME.

In section 6.3 we demonstrate the interaction with another framework by requesting information about devices and services from UPNP devices using our framework.

Next we evaluate our work by comparing the fulfillment of the identified requirements by related frameworks and our solution in section 6.4. Later on we will point out critical points of our solution and make a summary of the evaluation.

6.1 Message Definition and Encoding

Here we show the size of the messages represented in A3ME QL, their representation in the ASN.1 syntax and the encoded byte-stream size in ASN.1 PER encoding.

The possibility to transmit different types and combinations of structured information corresponds to one part of our goal of building a generic solution (1.2.3). In contrast most solutions in resource-constrained environments like for example WSN use static formats for messages without any context information, making it impossible to verify or interpret the content without external context.

Additionally all messages in our framework are based on our ASN.1 based message and content definitions. This allows to verify all our messages while being in ASN.1 format according to the definition. All A3ME messages used in this evaluation are automatically evaluated in our prototypical implementation before encoding. This possibility to verify a message of being compliant to a given schema is very important in the machine to machine (m2m) communication.

Queries Q1, Q2, Q3, Q4 show example requests defined in A3ME-QL.

```
REQUEST device.type-code, device.name
```

Listing 31: A3ME Query Q1 for Devices

```
REQUEST capability.id-nr, capability.type-code
```

Listing 32: A3ME Query Q2 for all Capabilities of Devices.

```
REQUEST voltage.type-code, voltage.data
PERIOD 10 second
```

Listing 33: A3ME Query Q3 for Battery Voltage every 10 Seconds.

```
REQUEST voltage.type-code, voltage.data,
PERIOD 1 minute
```

Listing 34: A3ME Query Q4 for Battery Voltage every Minute.

Listing 35 shows the representation of the query Q2 in ASN.1 syntax. The message in the listing additionally contains the conversation id of the message: *(ws:neo-mb,12)*.

```
Message ::= {
  performative request,
  content request-content : {
    what data-columns : {
      {a3me-code capability, infotype id-nr},
      {a3me-code capability, infotype type-code}
    }
  },
  conversation-id {owner "ws:neo-mb", conversationNr 12}
}
```

Listing 35: ASN.1 Representation of the A3ME Query Q2.

The ASN.1 representation of the query Q2 is encoded into byte stream using ASN.1 packed encoding rule (PER) and results in a payload of 20 bytes (see listing 36). This efficiently encoded payload can easily be transmitted using various communication techniques including those used in the resource-constrained WSNs.

```
10 84 00 00 10 7a 07 84 00 27
bf 37 5b b2 ef 5b b7 10 08 60
```

Listing 36: Binary Representation of the A3ME Query Q2.

The answer message (listing 37) from a sunspot device contains a result set with 2 columns and 12 rows. When represented in ASN.1 as text it requires about 887 bytes (without spaces and new lines) and when encoded using ASN.1 PER it results in 202 bytes payload. Strings like device names and addresses are not compressed, but numbers and enumerations can be very efficiently encoded and result in much smaller byte payload. The possibility of encoding the information in a very compact way verifies the requirement R5 *"Low Hardware Requirements."* (2.5) by reducing the amount of data needed to be transmitted.

```
Message ::= {
  performative inform,
  sender {
    name "sunspot:251F"
  },
  content inform-content : {
    sequence-number 0,
    resultset {
      schema {
        {a3me-code capability, infotype id-nr}, {a3me-code capability, infotype type-code}
      },
      rows {
        {integer-data : 12, integer-data : 54},
        {integer-data : 11, integer-data : 51},
        {integer-data : 10, integer-data : 50},
        {integer-data : 9, integer-data : 54},
        {integer-data : 8, integer-data : 54},
        {integer-data : 7, integer-data : 55},
        {integer-data : 6, integer-data : 44},
        {integer-data : 5, integer-data : 32},
        {integer-data : 4, integer-data : 21},
        {integer-data : 3, integer-data : 18},
        {integer-data : 2, integer-data : 19},
        {integer-data : 1, integer-data : 39}
      }
    }
  },
  conversation-id {owner "sunspot:251F", conversationNr 3},
  in-reply-to {owner "ws:neo-mb", conversationNr 12},
  received-from {address-type "EUI-64", address "0014.4F01.0000.251F"}
}
```

Listing 37: ASN.1 Representation of the A3ME Answer Message from a Sun Spot to the Query Q2.

As can be seen in Listing 35 and Listing 37 the messages used in the A3ME framework are well structured, self-descriptive and can be encoded to very compact byte streams.

6.2 Experiments and Measurements using A3ME Framework

In this section we describe different experiments and their results to demonstrate specific aspects of our framework. First we describe the different devices used in the experiments (Section 6.2.1). Next we describe experiments which demonstrate different aspects of the A3ME Framework:

- In Experiment 1 (Section 6.2.2) we demonstrate the interaction with different WSNs in a single query.
- In Experiment 2 (Section 6.2.3) we demonstrates interaction with heterogeneous devices and use of generic requests without requiring adjustments for interaction with the individual device types.
- In Experiment 3 (Section 6.2.4) we demonstrate the capability of our framework to run for longer time period and also to deal with individual nodes failing, restarting and reacquire the running query to continue sending the requested information.
- In Experiment 4 (Section 6.2.5) we demonstrates another use of the framework for a longer monitoring and a larger number of sensor devices.
- in Experiment 5 (Section 6.2.6) we demonstrate the ad-hoc discovery of devices and their capabilities from a smart phone across different devices.

6.2.1 Description of the Devices used for the Experiments

As the proof of concept and demonstration we use the prototypical implementations described in section 5. For the evaluation we use the following devices:

- Workstation MacBookPro4,1:
  - **OS:** Mac OS X 10.6.8,
  - **CPU:** Intel Core 2 Duo, 2 x 2,4 GHz,
  - **RAM:** DDR2 SDRAM 2 x 2 GB,
  - attached SunSpot base-station,
  - attached TelosB base-station.
- Smartphone Samsung Nexus S (section 5.4.1).
- Sun Spot sensor nodes with basic sensor boards[49] (section 5.2.1).
- MoteIV's TelosB sensor nodes (section 5.6.1).
- Zolertia sensor nodes (Z1) (section 5.7.1).

The Sun SPOTs and the TelosB together with Z1 build two wireless sensor networks (WSNs). TelosB and Z1 nodes are compatible to each other and build a common network. All three device types use the IEEE 802.15.4 compliant radio transceiver but while TelosB and Z1 nodes are compatible to each other, they are not compatible on the protocol layer above the physical layer to the SunSPOTs. Therefore the device types TelosB/Zolertia and SunSPOT are considered to be using different communication technique.

The nodes of the three device types are very different in terms of computing power and storage capabilities as shown in the table 11.

	Sun SPOT (rev6)	**MoteIV's TelosB**	**Zolertia Z1**
Processor	ARM9 processor	Texas Instruments MSP430 F1611 microprocessor	Texas Instruments MSP430 F2617 microprocessor
Computing power	180 MHz 32-bit	8 MHz 16-bit	16 MHz 16-bit
RAM memory	512 kB	10 kB	8 kB RAM
Flash memory	4096 kB	48 kB	92 kB
Radio transceiver	CC2420 2.4GHz IEEE 802.15.4 compliant	CC2420 2.4GHz IEEE 802.15.4 compliant	CC2420 2.4GHz IEEE 802.15.4 compliant
Power supply	build-in rechargeable battery, via USB cable	2 AA batteries, via USB cable	2 AA batteries, via USB cable
Leds	8 three-color leds	3 (red, green, blue)	3 (red, green, blue)
Button switches usable by applications	2	1	1
Temperature sensor	yes	yes	yes
Humidity sensor	–	yes	–
Light sensor	visible light	1 for visible light, 1 for visible and infra red light	–
Acceleration sensor	3-axis 2G/6G	–	3-axis, 2/4/8/16 G

Table 11: Comparison of the Technical Details of the Sun SPOT and TelosB Sensor Nodes.

The workstation and the smart phone are used as a user interface to visualize the information and to formulate and trigger requests or commands. Both interfaces provide predefined queries and the GUI on the workstation also allows to formulate or modify own queries.

In these scenario three communication technologies are used:

- [Com-1] Bluetooth between the smartphone and the workstation.
- [Com-2] IEEE 802.15.4 compliant radio between the TelosB sensor nodes and the base-station.
- [Com-3] IEEE 802.15.4 compliant radio between the Sun Spot sensor nodes and the base-station.

The workstation bridges the messages between these three communication interfaces if the messages are addressed to devices on the other interface or to all.

[49] Sun SPOT hardware revision 6.

6.2.2 Experiment 1: Interaction with Different WSNs in a Single Query

In this experiment we demonstrate the capability of our framework to request the information of two different WSNs by using a single query. The query is automatically forwarded by the workstation-device-agent to the available communication interfaces: TelosB and SunSPOT com-interfaces. Without the A3ME framework this would usually be done in two separate requests for each query.

We use the queries Q1 (Listing 31) and Q2 (Listing 32) to measure the response time from two different sensor device types: TelosB and SunSPOT. As described in section 6.2.1 the two device types are not compatible on the protocol layer above the physical layer and represent therefore two different independent WSNs.

The query Q1 asks for the device type and name and is usually used to find out which devices are available via A3ME framework. This will result in short one row answer per device. The query Q2 requests all capabilities of the devices, which will result in a multi row result-set for each device and therefore increasing the size of the answer messages compared to the query Q1.

Device name	**Hop distance**	**Response time in milliseconds**	
		Q1	**Q2**
sunspot:3C33	1	396	946
sunspot:4269	1	633	946
sunspot:4275	2	2026	1638
tmote-2	1	2822	3366
tmote-6	1	3196	3978

Table 12: Example Response Times for Queries Q1 and Q2.

Table 12 shows example response times for three sunspots and two TelosB sensor nodes. The response times for Q2 are considerably longer for both types of devices. This can be explained with the longer processing time to decode the request, generate the result and encode the answer message. The high variance of the measurements e.g. for the 1-hop connected Sun Spots results from wait times till the communication medium gets free to send.

6.2.3 Experiment 2: Use of Generic Requests and a Periodical Query

In this experiment we demonstrate a typical monitoring query which requests sensor readings periodically. The difference by using our framework is that a single query is used to collect sensor readings from different devices without knowing them and their sensors in detail. The information requested is just specified by the classification *sensor:voltage* and the infotypes *type-code* and *data*. This validates the capability of the A3ME framework to interact with heterogeneous devices and using generic requests without requiring adjustments for the individual device types. The use of the keyword "*PERIOD*" demonstrates the use of a periodical query, which triggers the recipients to send their results periodically.

For evaluation we use query Q3 in listing 33 to request sensor readings for battery voltage every 10 seconds.

Device name	**Active time**	**Messages count**		**Messages per minute**	
	(minutes)	**total**	**unique**	**total**	**unique**
sunspot:251F	314	3387	504	10.8	1.6
sunspot:3C33	481	7499	2697	15.6	5.6
sunspot:4269	482	7525	2616	15.6	5.4
sunspot:4275	481	6910	2391	14.3	5.0
tmote-2	13	62	19	4.8	1.5
tmote-6	477	3611	1199	7.6	2.5

Table 13: Amount of Messages from Each Device for the Query Q3 Running for 8 Hours.

Figure 34 shows the voltage decrease measured over time for 8 hours on battery powered sensor nodes. The query was started from the workstation and the workstation monitored all outgoing and incoming messages.

Table 13 shows the amount of messages received from the different nodes in total and per minute. As we can see in the table and in Figure 34 "tmote-2" and "sunspot:251F" were not functioning properly. "tmote-2" stopped responding after 13 minutes and "sunspot:251F" also stopped responding after some time and was then restarted (manually). The amount of sensor readings of the remaining Sun Spot devices was with 5.3 unique messages per minute in average close

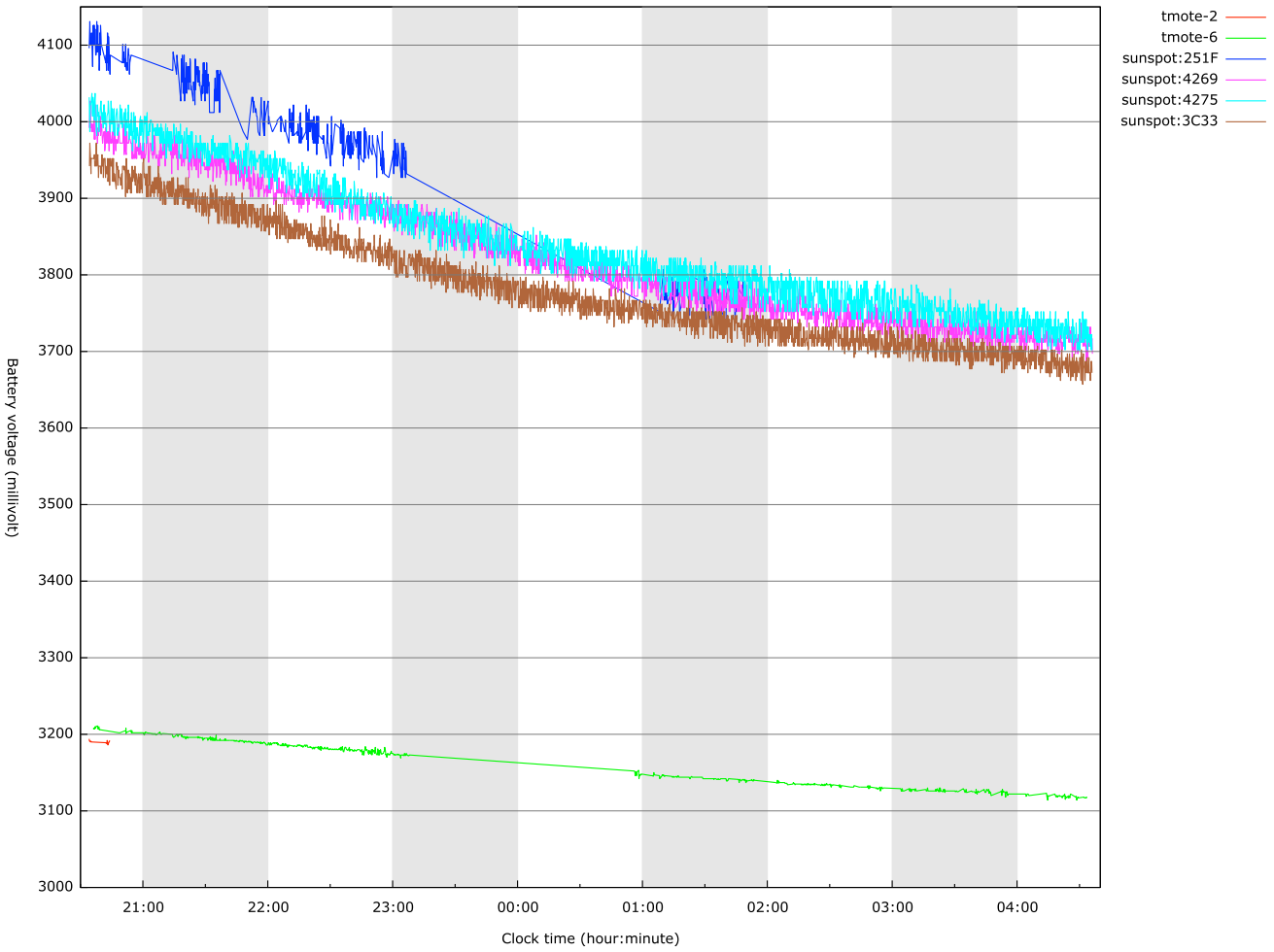


Figure 34: 8 Hour Voltage Measurements on TelosB and Sun Spot Sensor Nodes

to optimal value of 6 messages per minute (the query requested sensor readings every 10 seconds). The messages from ”tmote-6“ arrived only with the rate of 2.5 messages per minute. Here more than half of the messages were lost due to less reliable communication hardware and collisions with Sun Spots transmissions.

6.2.4 Experiment 3: Long Running Query and Dealing with Individual Nodes Failing and Restarting

This experiment demonstrates the capability of our framework to run for longer time period and also to deal with individual nodes failing, restarting and reacquire the running query to continue sending the requested information (see figure 35).

In this experiment we again use query Q3 in listing 33 to request sensor readings for battery voltage every 10 seconds. We monitor the timestamps for the reception of each message to see when the devices stop sending their messages. This enables us to measure the life time of the sensor nodes till they run out of energy.

All sensor nodes in this experiment are powered by batteries:

- Sun Spots have a build-in Lithium-ion battery with 3.7V 720mAh (see 5.2.1),
- Tmote and Z1 sensor nodes are powered by 2 AA batteries with 1.5V 2800mAh (see 5.6.1).

Device	Battery	Energy (mWh)
Sun Spot	rechargeable LI-ION battery	2664
TelosB (tmotes)	2x AA alkaline batteries	8400
Z1	2x AA alkaline batteries	8400

Table 14: Amount of Energy per Sensor Node.

For energy consumption we use the amount of received messages from each node. The table 14 shows the energy amount available for the sensor nodes. While the energy amount available for the individual TelosB and Z1 sensor nodes is equal for all nodes – we equipped all nodes with new AA batteries of same type and brand, on the Sun Spots the real amount of the available energy might vary dependent on the age and status of the rechargeable battery. The maximum energy charge here might be reduced.

Device name	Life time (in minutes)	Message count	Energy consumption (in μWh)	
			per minute	per msg
sunspot:3664	245	1243	10873	2143
sunspot:396C	773	4205	3446	634
sunspot:3C02	754	4263	3533	625
sunspot:3C33	752	3953	3543	674
sunspot:3D1D	334	1754	7976	1519
sunspot:4269	381	2006	6992	1328
sunspot:4275	399	2206	6677	1208
tmote-6	3160	11535	2658	728
z1-2	3206	10136	2620	829
z1-201	3159	16501	2659	509
z1-214	2021	7633	4156	1100

Table 15: Lifetime and Average Energy Consumption for Each Device for the Query Q3.

Figure 35 shows the sensor readings collected over 24 hours till no further sensor readings were received. The amount of energy available on each sensor node and the total number of messages received from each device enables us to calculate the energy per minute and the energy per received message. Table 15 shows the energy consumption for query Q3 for each device and the figure 36 shows the corresponding diagrams.

This experiment shows a surprising result: according to our measurements the Sun Spot sensor nodes have consumed less energy per message and per minute. The expected result would be that the TelosB and Z1 sensor nodes are more energy efficient since they are equipped with the low power MSP430 microprocessor, while the Sun Spots use a more powerful ARM9 processor. This can be explained with the fact that more messages get lost for the TelosB and Z1 nodes and that the high query frequency of 1 message every 10 seconds did not allow the TelosB and Z1 nodes to go into deep sleep to save energy. The surprising results observed in this experiment could only be achieved by querying three different sensor platforms in one common framework (A3ME).

Another observation from this experiment is the significantly higher ratio of delivered messages from Sun Spots than from TelosB and Z1 nodes compared to the amount of messages expected in the given time (Figure 37).

6.2.5 Experiment 4: Long Running Query on a Larger Number of Sensor Devices.

To demonstrate the A3ME framework for sensor monitoring we run a continuous query asking for light (illuminance) sensor readings on two different sensor networks simultaneously. The first wireless sensor network is the testbed TUDμNet[50] [83] [84]. It is a wireless sensor network testbed developed at Technische Universität Darmstadt, which allows researchers to test and evaluate algorithms and protocols on a real sensor network. The second sensor network is represented by a set of SunSpot devices.

For our evaluation we used the TelosB sensor Network of the TUDμNet deployed in the computer science building. We did not any special optimization or adaptation for the evaluation run. The Sun Spots are deployed additionally in the office where the workstation triggering the query is located.

For this evaluation run we decided to query the light measurements. We request the sensor measurements for light intensity (illuminance) in lux every 30 minutes without any other restrictions in the query. Listing 38 shows the representation of the query Q5 in A3ME-QL syntax.

```
REQUEST light.type-code, light.data
PERIOD 30 second
```

Listing 38: A3ME Query Q5 for Light (Illuminance) Sensor Readings Every 30 Second.

[50] TUDμNet web page: http://tudunet.dvs.informatik.tu-darmstadt.de

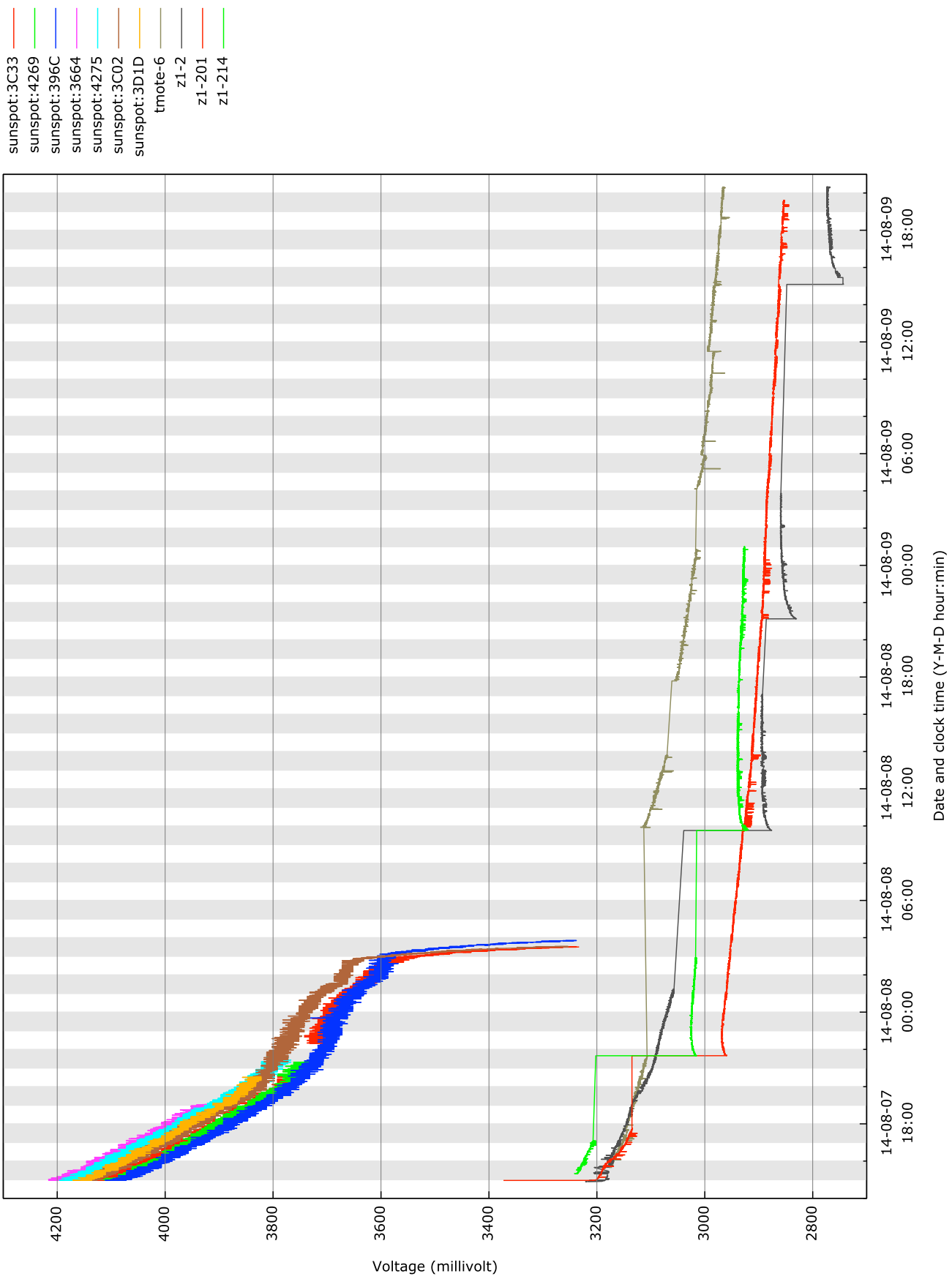


Figure 35: 24 Hour Voltage Measurements on TelosB and SunSpot Sensor Nodes

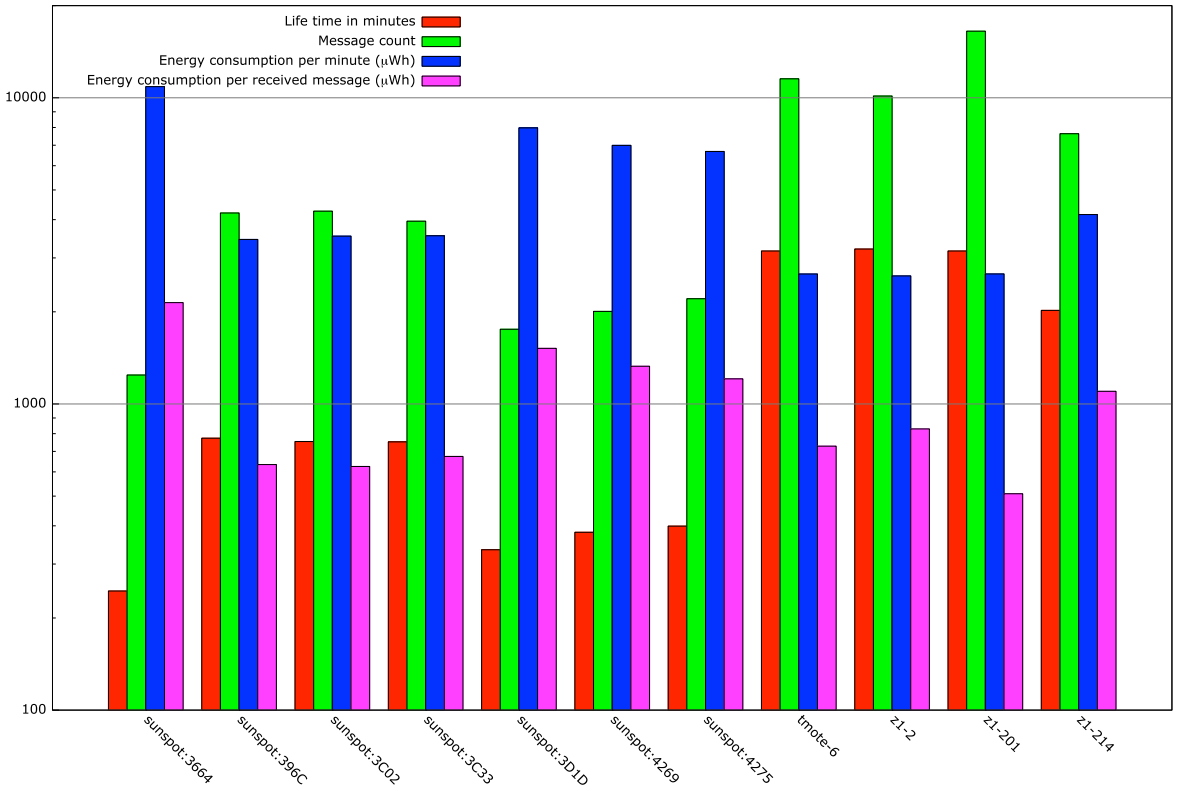


Figure 36: Comparison of Lifetime and Average Energy Consumption.

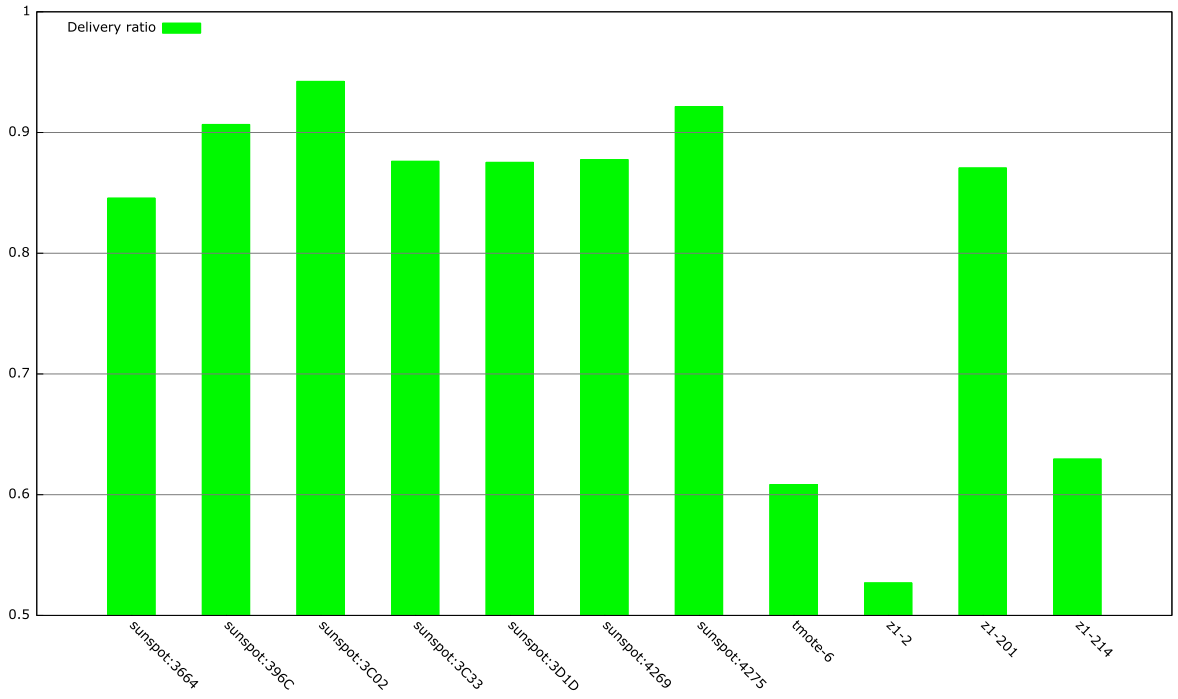


Figure 37: Message Delivery Ratio from Different Sensor Nodes.

38 sensor nodes were programmed with the A3ME program image via the TUDμNet. Additionally to the TelosB sensor nodes from TUDμNet we deployed 3 Sun Spot devices to participate in the evaluation run. The query was formulated and triggered from a workstation equipped with a TelosB and SunSpot base stations to communicate with the two sensor networks. The workstation collected all incoming and outgoing messages. Additionally the TUDμNet stored all local text outputs of the nodes in a relational database.

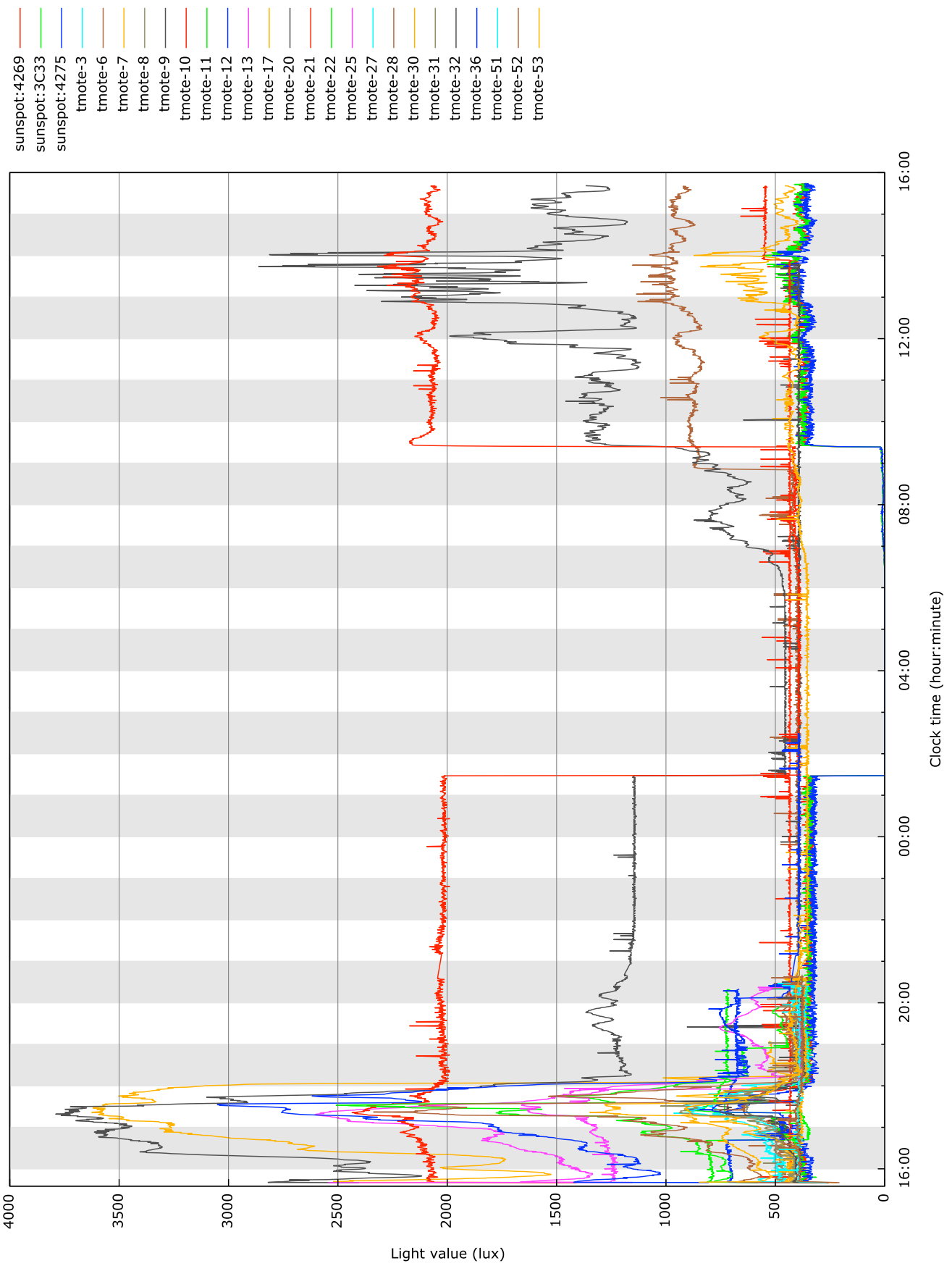


Figure 38: 24 Hour Light Measurements on the TUDμNet and Sun Spots

The query was running for 24 hours started at 15:35 on 29.07.2014 and stopped on 15:40 of the next day. We added 5 minutes to the evaluation time to compensate the time TUDμNet requires to program and restart the sensor nodes. Figure 38 shows the result of the 24 hour evaluation run. The Figure shows the light measurements of 3 Sun Spots and 23 TelosB sensor nodes. We did not get any response from the remaining 15 TelosB sensor nodes either because the malfunction of the nodes themselves or because they were disconnected from the rest of the network through the failure of the other nodes via which they would be connected.

During the evaluation run the workstation monitored 35068 unique messages. 249 messages were hello messages send out by nodes on their startup. This means that some of the sensor nodes restarted them selves at some point in time. This fact does not hurt since these nodes got the query with the next query-refresh message and continued then to send the sensor readings as requested.

During the run 722 refresh messages were send: one refresh message every 2 minutes. The remaining 34097 messages were sensor readings received from the 26 sensor nodes (3 Sun Spots and 23 TelosB). This results in 1311 messages per sensor node in average and corresponds to 0.91 message per sensor node per minute. With the optimal expected amount of 2 messages per device per minute this means that 45.5 percent of messages were delivered.

The light peak from 16:20 till 18:00 o'clock indicates the light of the lower standing evening sun shining directly in some of the offices where the sensors were deployed. At 01:30 o'clock we also can see the light being switched off and switched on again at about 09:25 o'clock in one of the rooms.

This evaluation run demonstrates the use of the A3ME framework to access and to collect data from two very different sensor networks via common interface.

6.2.6 Experiment 5: Query for Devices and their Capabilities

In this experiment we run the query Q1 (Listing 31) to discover all devices reachable by the A3ME framework in the computer science building of the Technische Universität Darmstadt and query Q2 (Listing 32) to collect the capabilities of the devices. The query is started from the Samsung 'Nexus S" smart phone and asks for the device types and names. this experiment demonstrates the possibility of ad-hoc device discovery and the collection of information about devices capabilities available at the actual position of the querying device.

The query formulated in A3ME-QL is translated into ASN.1 syntax and encoded using ASN.1 PER encoding. The encoded message is transmitted to all connected Bluetooth devices (here the other smartphone and the workstation). The receiving workstation forwards the message to the other smartphone via Bluetooth interface, to the TelosB and Zolertia nodes via the TelosB interface and to the SunSpot nodes via the SunSpot interface. The other smartphone forwards the request to the workstation and the sensor nodes forward the message to other sensor nodes. After forwarding the message the devices evaluate and answer the query themselves by sending the answer to the interface where the original request came from.

Table 16 shows the result for the query Q1. The number "7" is the A3ME-code from the A3ME classification and corresponds to a "mote device', number "8" corresponds to a "mobile device" and number "5" corresponds to a "device". On Figure 39a we also see the visualization of the collected information on the smart phone. We got responses from

- 1 another smartphone "android:Nexus 4",
- 6 Sun Spot sensor nodes,
- 39 tmote TelosB sensor nodes (deployed via TUDμNet),
- 6 Zolertia Z1 sensor nodes and
- 1 workstation "ws:kailua".

Next we start the query Q2 (Listing 32) to collect the information about capabilities of the available devices. The information of the responses is integrated into the device-info view on the smart phone (Figure 39b).

This experiment demonstrated the capability to collect information about other devices available in the proximity of a device. In this case in the proximity of the smart phone. Furthermore we collected the information about the capabilities of the devices. The information collected was not only originated from devices capable to communicate with the smart phone directly but also indirectly via another device (here the workstation "ws:kailua") which bridged the communication to and from the three different communication networks.

Next step for the future work here would be to enhance the graphical user interfaces of the workstation and the smartphone further, to allow the user to select devices, capabilities, services, etc. and to request the information on selection and present it to the user as soon as the answers arrive.

6.3 Exemplary Bridging to the UPNP Framework

To demonstrate the interaction with the UPNP framework, as described in section 4.5.6 and implemented in 5.3.5, we query the devices and the services from the A3ME-workstation which is connected to a local network with other UPNP devices. For this evaluation we executed the following queries in the Smart Home Laboratory of the VDE Testing and Certification Institute, where a broad variety of devices is present in the local network.

8, android:Nexus 4	7, tmote-2	7, tmote-45
7, sunspot:3664	7, tmote-20	7, tmote-47
7, sunspot:396C	7, tmote-21	7, tmote-5
7, sunspot:3C02	7, tmote-22	7, tmote-50
7, sunspot:3C33	7, tmote-23	7, tmote-51
7, sunspot:3D1D	7, tmote-25	7, tmote-52
7, sunspot:4275	7, tmote-27	7, tmote-53
7, tmote-1	7, tmote-28	7, tmote-6
7, tmote-10	7, tmote-3	7, tmote-7
7, tmote-11	7, tmote-30	7, tmote-8
7, tmote-12	7, tmote-31	7, tmote-9
7, tmote-13	7, tmote-33	5, ws:kailua
7, tmote-14	7, tmote-35	7, z1-1
7, tmote-15	7, tmote-36	7, z1-12
7, tmote-16	7, tmote-37	7, z1-201
7, tmote-17	7, tmote-38	7, z1-214
7, tmote-18	7, tmote-39	7, z1-215
7, tmote-19	7, tmote-4	7, z1-216

Table 16: Result of the Query Q1.

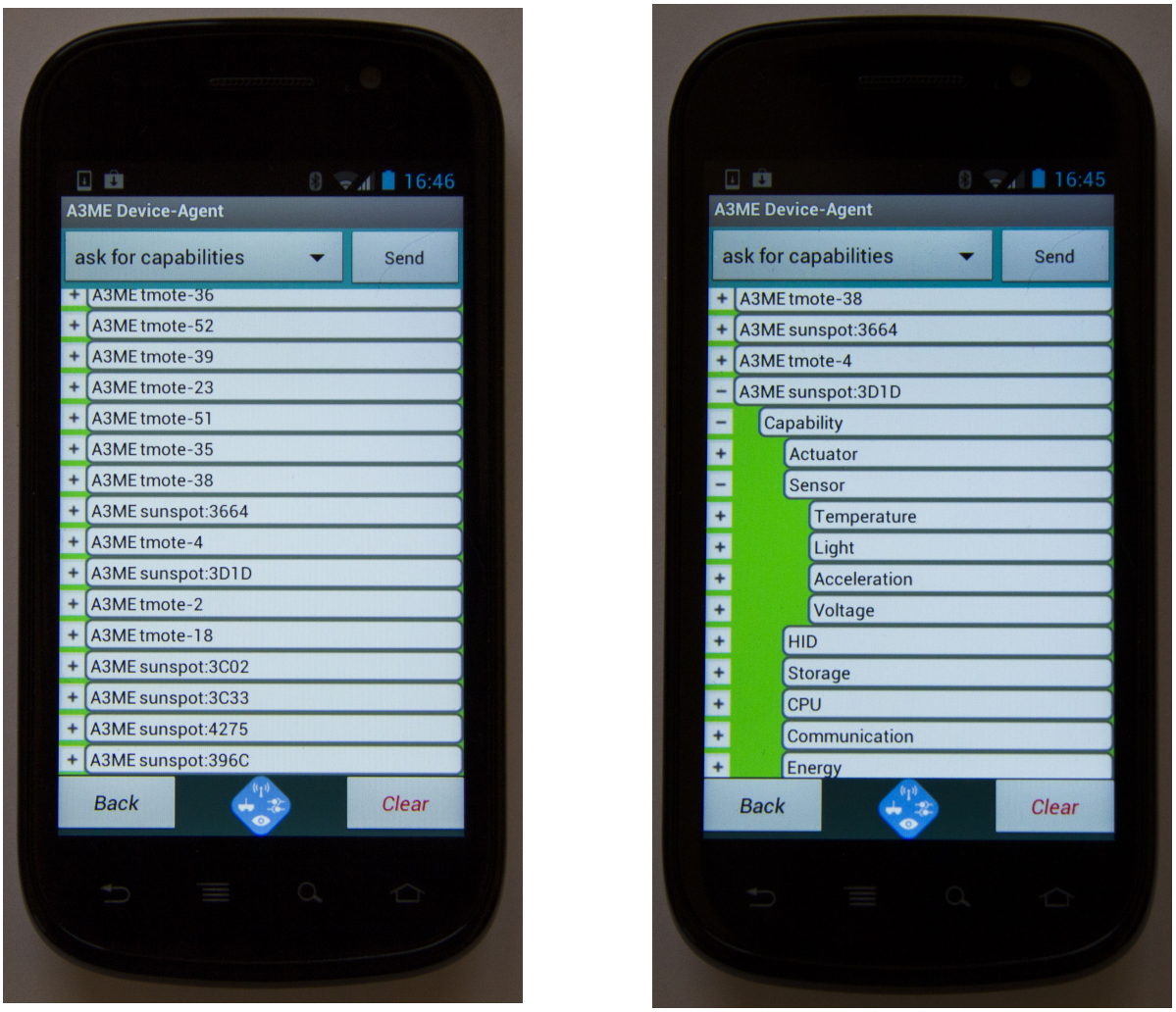
In query Q6 we request devices with their name, id, type, description and m2m-description.

```
REQUEST device.name, device.data, device.id-nr, device.description, device.m2m-description
```

Listing 39: A3ME Query Q6 for Devices

The listing 40 shows the replies to the query Q6.

```
Answer: from (UPNP,1) 1
(Device.name), (Device.data), (Device.id-nr), (Device.description), (Device.m2m_description)
-------------------------
Samsung Electronics UE55ES8090 1.0,
[TV]UE55ESVDE,
uuid:0f7f4900-0004-1000-9450-f47b5e0cc0d3,
Type=urn:samsung.com:device:RemoteControlReceiver:1, Manufacturer=Samsung Electronics,
http://192.168.178.72:7676/smp_2_
-------------------------
Arthur Herzog a3me-da-workstation v1,
A3ME workstation device-agent,
uuid:ab2f6615-c228-4db4-ffff-ffffaa1d8c04,
Type=urn:schemas-upnp-org:device:A3ME:1, Manufacturer=Arthur Herzog,
http://192.168.178.74:49196/dev/ab2f6615-c228-4db4-ffff-ffffaa1d8c04/desc
-------------------------
Samsung Electronics SEC001599B5C7DD 1.0,
Samsung CLX-3300 Series (192.168.178.71),
uuid:16a65700-007c-1000-bb49-001599b5c7dd,
Type=urn:schemas-upnp-org:device:Printer:1, Manufacturer=Samsung Electronics,
http://192.168.178.71:5200/Printer.xml
-------------------------
Sonos, Inc. Sonos PLAY:1 S1,
192.168.178.59 - Sonos PLAY:1,
uuid:RINCON_B8E9378420FA01400,
Type=urn:schemas-upnp-org:device:ZonePlayer:1, Manufacturer=Sonos, Inc.,
http://192.168.178.59:1400/xml/device_description.xml
-------------------------
Busch-Jaeger Elektro System Access Point 6200 AP,
System-Access-Point,
uuid:658e0e00-050e-7f65-000e-7f650e658e0e,
Type=urn:busch-jaeger:device:SysAP:1, Manufacturer=Busch-Jaeger Elektro,
http://192.168.178.84:80/sysap.upnp
-------------------------
```



(a) Devices Available in the Proximity of the Smart Phone

(b) Capabilities Information from the Sun Spot "sunspot:3D1D"

Figure 39: Smartphone with the Collected Information:

```
Samsung Electronics UE55ES8090 1.0,
[TV]UE55ESVDE,
uuid:08583b00-008c-1000-b49e-f47b5e0cc0d3,
Type=urn:dial-multiscreen-org:device:dialreceiver:1, Manufacturer=Samsung Electronics,
http://192.168.178.72:7676/smp_6_
-----------------------------------
Miele & Cie. KG XGW3000,
XGW3000,
uuid:1cc50d74-e274-11e4-9fba-0025dc69b8db,
Type=urn:schemas-upnp-org:device:basic:1, Manufacturer=Miele & Cie. KG,
http://192.168.178.90:49152/upnpbd4.xml
-----------------------------------
Microsoft Corporation Windows Media Player Sharing 12.0,
SMARTHOME: SmartHomeUser:,
uuid:41d0c1d8-8ec1-4175-9322-1387f0d60337,
Type=urn:schemas-upnp-org:device:MediaServer:1, Manufacturer=Microsoft Corporation,
http://192.168.178.95:2869/upnphost/udhisapi.dll?content=uuid:41d0c1d8-8ec1-4175-9322-1387f0d60337
-----------------------------------
AVM Berlin FRITZ!Box 3390 avm,
dss-fb Mediaserver,
uuid:fa095ecc-e13e-40e7-8e6c-3431c4034ca1,
Type=urn:schemas-upnp-org:device:MediaServer:1, Manufacturer=AVM Berlin,
http://192.168.178.10:49000/MediaServerDevDesc.xml
-----------------------------------
D-Link DAP-1522 00000000,
```

```
DAP-1522,
uuid:40ab5dfb-d314-36cc-19e0-c8d3a3531e12,
Type=urn:schemas-wifialliance-org:device:WFADevice:1, Manufacturer=D-Link,
http://192.168.178.9:45555/wps_device.xml
-------------------------
Microsoft Corporation Windows Media Player Sharing 12.0,
SMARTHOME: Administrator:,
uuid:7d247eca-aa94-4feb-a882-edb2ad74df19,
Type=urn:schemas-upnp-org:device:MediaServer:1, Manufacturer=Microsoft Corporation,
http://192.168.178.95:2869/upnphost/udhisapi.dll?content=uuid:7d247eca-aa94-4feb-a882-edb2ad74df19
-------------------------
Deutsche Telekom AG QIVICON,
qivicon,
uuid:baaeb720-d41c-11e0-85e8-0025dc68c27f,
Type=urn:schemas-upnp-org:device:basic:1, Manufacturer=Deutsche Telekom AG,
http://192.168.178.81:49152/upnpbd4.xml
-------------------------
AVM Berlin FRITZ!Box 3390 avm,
dss-fb,
uuid:75802409-bccb-40e7-8e6c-3431C4034CA1,
Type=urn:schemas-upnp-org:device:InternetGatewayDevice:1, Manufacturer=AVM Berlin,
http://192.168.178.10:49000/igddesc.xml
-------------------------
Samsung Electronics UE55ES8090 1.0,
[TV]UE55ESVDE,
uuid:1017df80-000e-1000-98fb-f47b5e0cc0d3,
Type=urn:samsung.com:device:MainTVServer2:1, Manufacturer=Samsung Electronics,
http://192.168.178.72:7676/smp_10_
-------------------------
Samsung Electronics UE55ES8090 AllShare1.0,
[TV]UE55ESVDE,
uuid:10b07600-0018-1000-ae4b-f47b5e0cc0d3,
Type=urn:schemas-upnp-org:device:MediaRenderer:1, Manufacturer=Samsung Electronics,
http://192.168.178.72:7676/smp_18_,
```

Listing 40: Answers for the Query Q6

In query Q7 we request services with their hosting device name, service name, description and m2m-description.

```
REQUEST device.name, service.name, service.description, service.m2m-description
```

Listing 41: A3ME Query Q7 for Services

The listing 42 shows the replies to the query Q7.

```
Answer: from (UPNP,2) 1
(Device.name), (Service.name), (Service.description), (Service.m2m_description),
-------------------------
Samsung Electronics UE55ES8090 1.0,
urn:samsung.com:serviceId:MultiScreenService,
Type=urn:samsung.com:service:MultiScreenService:1,
http://192.168.178.72:7676/smp_3_
-------------------------
Sonos, Inc. Sonos PLAY:1 S1,
urn:upnp-org:serviceId:AlarmClock,
Type=urn:schemas-upnp-org:service:AlarmClock:1,
http://192.168.178.59:1400/xml/AlarmClock1.xml
-------------------------
Sonos, Inc. Sonos PLAY:1 S1,
urn:upnp-org:serviceId:MusicServices,
Type=urn:schemas-upnp-org:service:MusicServices:1,
http://192.168.178.59:1400/xml/MusicServices1.xml
-------------------------
Sonos, Inc. Sonos PLAY:1 S1,
urn:upnp-org:serviceId:DeviceProperties,
Type=urn:schemas-upnp-org:service:DeviceProperties:1,
http://192.168.178.59:1400/xml/DeviceProperties1.xml
-------------------------
Sonos, Inc. Sonos PLAY:1 S1,
urn:upnp-org:serviceId:SystemProperties,
Type=urn:schemas-upnp-org:service:SystemProperties:1,
http://192.168.178.59:1400/xml/SystemProperties1.xml
```

```
-----------------------------
Sonos, Inc. Sonos PLAY:1 S1,
urn:upnp-org:serviceId:ZoneGroupTopology,
Type=urn:schemas-upnp-org:service:ZoneGroupTopology:1,
http://192.168.178.59:1400/xml/ZoneGroupTopology1.xml
-----------------------------
Sonos, Inc. Sonos PLAY:1 S1,
urn:upnp-org:serviceId:GroupManagement,
Type=urn:schemas-upnp-org:service:GroupManagement:1,
http://192.168.178.59:1400/xml/GroupManagement1.xml
-----------------------------
Sonos, Inc. Sonos PLAY:1 S1,
urn:tencent-com:serviceId:QPlay,
Type=urn:schemas-tencent-com:service:QPlay:1,
http://192.168.178.59:1400/xml/QPlay1.xml
-----------------------------
Samsung Electronics UE55ES8090 1.0,
urn:dial-multiscreen-org:serviceId:dial,
Type=urn:dial-multiscreen-org:service:dial:1,
http://192.168.178.72:7676/smp_7_
-----------------------------
Microsoft Corporation Windows Media Player Sharing 12.0,
urn:upnp-org:serviceId:ConnectionManager,
Type=urn:schemas-upnp-org:service:ConnectionManager:1,
http://192.168.178.95:2869/upnphost/udhisapi.dll?content=uuid:46940ce6-5003-4a8a-837a-08d3d52114f2
-----------------------------
Microsoft Corporation Windows Media Player Sharing 12.0,
urn:upnp-org:serviceId:ContentDirectory,
Type=urn:schemas-upnp-org:service:ContentDirectory:1,
http://192.168.178.95:2869/upnphost/udhisapi.dll?content=uuid:430af187-c947-410e-b720-ef333a90669a
-----------------------------
Microsoft Corporation Windows Media Player Sharing 12.0,
urn:microsoft.com:serviceId:X_MS_MediaReceiverRegistrar,
Type=urn:microsoft.com:service:X_MS_MediaReceiverRegistrar:1,
http://192.168.178.95:2869/upnphost/udhisapi.dll?content=uuid:364cdb5f-6410-4d89-b279-b77d231f47f6
-----------------------------
AVM Berlin FRITZ!Box 3390 avm,
urn:upnp-org:serviceId:ContentDirectory,
Type=urn:schemas-upnp-org:service:ContentDirectory:1,
http://192.168.178.10:49000/MediaServerContentDirectory.xml
-----------------------------
AVM Berlin FRITZ!Box 3390 avm,
urn:upnp-org:serviceId:ConnectionManager,
Type=urn:schemas-upnp-org:service:ConnectionManager:1,
http://192.168.178.10:49000/MediaServerConnectionManager.xml
-----------------------------
AVM Berlin FRITZ!Box 3390 avm,
urn:microsoft.com:serviceId:X_MS_MediaReceiverRegistrar,
Type=urn:microsoft.com:service:X_MS_MediaReceiverRegistrar:1,
http://192.168.178.10:49000/MediaReceiverRegistrar.xml
-----------------------------
AVM Berlin FRITZ!Box 3390 avm,
urn:avm.de:serviceId:AVM_ServerStatus,
Type=urn:avm.de:service:AVM_ServerStatus:1,
http://192.168.178.10:49000/ServerStatus.xml
-----------------------------
D-Link DAP-1522 00000000,
urn:wifialliance-org:serviceId:WFAWLANConfig1,
Type=urn:schemas-wifialliance-org:service:WFAWLANConfig:1,
http://192.168.178.9:45555wps_scpd.xml
-----------------------------
Microsoft Corporation Windows Media Player Sharing 12.0,
urn:upnp-org:serviceId:ConnectionManager,
Type=urn:schemas-upnp-org:service:ConnectionManager:1,
http://192.168.178.95:2869/upnphost/udhisapi.dll?content=uuid:c9c285e6-6c55-4169-8bb0-2b6184eeb92c
-----------------------------
Microsoft Corporation Windows Media Player Sharing 12.0,
urn:upnp-org:serviceId:ContentDirectory,
Type=urn:schemas-upnp-org:service:ContentDirectory:1,
http://192.168.178.95:2869/upnphost/udhisapi.dll?content=uuid:814b67b4-7040-4a58-b600-02a98f42e10f
```

```
-----------------------------
Microsoft Corporation Windows Media Player Sharing 12.0,
urn:microsoft.com:serviceId:X_MS_MediaReceiverRegistrar,
Type=urn:microsoft.com:service:X_MS_MediaReceiverRegistrar:1,
http://192.168.178.95:2869/upnphost/udhisapi.dll?content=uuid:f75a7537-79ab-4f43-a0b2-514e6c2aa9a5
-----------------------------
AVM Berlin FRITZ!Box 3390 avm,
urn:any-com:serviceId:any1,
Type=urn:schemas-any-com:service:Any:1,
http://192.168.178.10:49000/any.xml
-----------------------------
Samsung Electronics UE55ES8090 1.0,
urn:samsung.com:serviceId:MainTVAgent2,
Type=urn:samsung.com:service:MainTVAgent2:1,
http://192.168.178.72:7676/smp_11_
-----------------------------
Samsung Electronics UE55ES8090 AllShare1.0,
urn:upnp-org:serviceId:RenderingControl,
Type=urn:schemas-upnp-org:service:RenderingControl:1,
http://192.168.178.72:7676/smp_19_
-----------------------------
Samsung Electronics UE55ES8090 AllShare1.0,
urn:upnp-org:serviceId:ConnectionManager,
Type=urn:schemas-upnp-org:service:ConnectionManager:1,
http://192.168.178.72:7676/smp_22_
-----------------------------
Samsung Electronics UE55ES8090 AllShare1.0,
urn:upnp-org:serviceId:AVTransport,
Type=urn:schemas-upnp-org:service:AVTransport:1,
http://192.168.178.72:7676/smp_25_
```

Listing 42: Answers for the Query Q7

The queries Q6 and Q7 demonstrate the interconnection of another framework (here UPNP) with the A3ME framework. This interconnection only required to implement and add a new communication-interface for UPNP as described in section 5.3.5. No other parts of the A3ME implementation and UPNP had to be changed or adjusted.

6.4 Requirements Fulfillment by Different Frameworks

In this section we will evaluate the frameworks by checking the fulfillment of the requirements defined in section 2.

The table 17 shows the consolidated list of the requirements defined in section 2 which also correspond to columns of the comparison table 18.

TinyDB: As described in section 3.1.2 TinyDB is a database-inspired approach for querying sensor readings from wireless sensor networks. It treats the whole WNS as an virtual database relation on which it allows to execute queries. The schema of the virtual relation must be known in advance, therefore it can not deal with heterogeneous devices with different kinds of sensors. In the basic solution the queries are defined on a workstation which communicates with the WSN through a connected base station sensor node.

This framework can only be applied to the WSN devices for which it provides an efficient solution with low hardware requirements.

Jini: The Jini framework (section 3.5.1) allows to offer and use Java based services among different devices in an IP based network (–> not communication technology independent). This means for the comparison that only the devices capable to run a Java VM can use it. For announcement and discovery of services a lookup service is used (–> centralized solution). This means that on-the-fly ad-hoc interactions without a lookup service are not possible. Device discovery and information query is not offered. Service discovery and offering is the major purpose of Jini.

Mundo: The Mundo framework (section 3.2.2) allows to assign and represent the different types of devices in our evaluation to the fife Mundo groups. The simple sensors and actuators are represented as IT (smart ITems). Smartphones would be the Mundo ME (Minimal Entity) and are representations of their users. The unmanned vehicles also best fit into the ME category, since they are entities which might be willing to interact with the environment similar to humans.

The workstations and servers belong to Mundo's US (Ubiquitous aSsociable objects) and are seen as providers of additional services like computing power, storage, etc.

Groups of these devices can be combined in Mundo's WE (Wireless group Environment). THEY (Telecooperative Hierarchical ovErlaY) offers external infrastructure to other MundoCore elements.

The ad-hoc interactions here are only possible between ME and other devices. To publish and discover services publish/subscribe mechanisms are used, what makes it a type of centralized solution, since the mechanism requires a broker as an intermediate device. Many different technologies can be used for communication.

R1	Self-description of devices (SD):
R1a	- Device classification,
R1b	- Capabilities classification,
R1c	- Classification extension,
R1d	- Implementation independent classification.
R2	Technology independent interaction (TII):
R2a	- Technology independent communication primitives,
R2b	- Flexible technology independent message structure,
R2c	- Common encoding and decoding schema for messages,
R2d	- Technology independent interaction protocols,
R2e	- Technology independent query language.
R3	Decentralized solution (DCS).
R4	Applicable to Heterogeneous Environments (Het):
R4a	- Applicable to WSN,
R4b	- Applicable to Ubiquitous Environments,
R4c	- Applicable to Unmanned Vehicles.
R5	Low hardware requirements (LHW).

Table 17: Condensed Requirements List.

The WSN devices are not really covered here, even-so it is possible to represent them as ITs. The Ubiquitous environments are the primary target area of MundoCore.

CORBA: The CORBA (section 3.5.2) is very flexible and technology independent middleware. It can be set up on all types of devices we consider to be present in MME. For the resource-constrained devices there is a CORBA for embedded devices, which is supposed to work even on one chip solutions.

The ad-hoc interactions and decentralized solution criteria are not satisfied by CORBA, since the solution requires a broker (ORB) to interact with other objects. CORBA also does not offer device, capabilities, information and status query about the devices, but its strength are the service discovery and execution.

UPNP: The UPNP framework (section 3.5.5) allows on-the-fly ad-hoc discovery of devices and their offered capabilities in IP based networks. The framework targets primarily residential networks like ubiquitous home environments. The framework does not rely on any central instance.

Currently UPnP does not support WSN nodes, but there is a working group working on it. UPnP has a low power Device Control Protocol which supports devices going into sleep modus. But nevertheless UPnP currently does not meet the low hardware requirement as described in 2.5.

AllJoyn: The AllJoyn framework (section 3.2.8) allows on-the-fly discovery of devices and their interactions with each other. One limitation is that resource constrained devices, called Leaf Nodes here, can not interact directly with other devices, but have to connect to a more powerful type of device (Router Node) and only then can interact with other devices. Therefore we rate the R3 Decentralized solution (DCS) criteria as only partially fulfilled. AllJoyn does not offer a classification of devices or capabilities (R1). The message exchange can be done using any of the communication technologies available on a device (R2a) and the messages can be constructed using a set of basic data types (R2b). For encoding/decoding messages for transport D-Bus [131] wire format is used.

Comparison Summary: The results of the evaluation are summarized in the table 18. In each cell we enter the rating of the given middleware (row) for the current criteria (column). The ratings are:

+	The criteria is fulfilled
o	The criteria is only partially fulfilled
–	The criteria is not fulfilled

6.5 Critical Points

Our solution requires each device to have the A3ME device-agent realization software installed to use the framework directly. Otherwise a device which does not have an A3ME device-agent realization software installed/available on it can only interact with the A3ME framework through another A3ME-enabled device.

	SD				TII					DCS	Het			LHW
	R1a	**R1b**	**R1c**	**R1d**	**R2a**	**R2b**	**R2c**	**R2d**	**R2e**	**R3**	**R4a**	**R4b**	**R4c**	**R5**
TinyDB	–	–	–	–	–	–	–	o	+	–	+	–	–	+
Jini	–	–	–	–	–	–	–	–	–	–	–	o	o	–
Mundo	+	–	–	–	+	+	+	–	–	–	–	+	–	–
Corba	–	–	–	–	+	+	+	+	o	–	+	+	+	+
UPNP	+	+	–	+	–	+	+	+	+	+	–	+	–	–
AllJoyn	–	–	–	–	+	+	+	+	–	o	+	+	+	+
A3ME	+	+	+	+	+	+	+	+	+	+	+	+	+	+

Table 18: Comparison of the Frameworks with Respect to the Requirements.

In A3ME we extensively use ASN.1 and ASN.1 Packed Encoding Rules (PER) (see section 4.13). This introduces a dependency from the ASN.1 encoding/decoding libraries. But since the ASN.1 and ASN.1 PER are open standards which are already used in many areas, there exist many different implementations to be used here.

6.6 Evaluation Summary

As described in section 4 and demonstrated in section 6 the A3ME framework satisfies the requirements defined in section 2:

R1 **Self-description of devices (SD):**
In A3ME each device can describe its device class and its capabilities using the A3ME classification (section 4.7). This classification can be extended on purpose and can be used independent of the programming languages and technologies used for implementation of the individual devices.

R1a **Device classification:**
In A3ME each device belongs to one class of predefined device types and can describe itself as such.

R1b **Capabilities classification:**
All capabilities are classified using the A3ME capabilities classification. This classification defines basic classes of capabilities: sensor, actuator, human interface, energy, information storage, communication and computing. Most capabilities can be described as subclass of one of these eight basic capability classes. For all capabilities which do not fit into one of these eight basic capability classes the 'other capability' class can be used. Some of the basic capability classes are further subclassified to cover often used capabilities.

R1c **Classification extension:**
In section 4.7.4 we described two ways to extend the predefined classification.

R1d **Implementation independent classification:**
The classification can be implemented using description languages like RDF and OWL and its corresponding mechanisms. Alternatively the classification can be also implemented as simple key-value pairs. Allowing the to use it on resource constrained devices.

R2 **Technology independent interaction (TII):**
The A3ME framework is designed that way, that different techniques and technologies can be used in parallel. The used techniques can also be replaced or extended by different ones on purpose or as consequence of the technical advance.

R2a **Technology independent communication primitives:**
Each device can have and use more than one communication technology, and its corresponding addressing schema, protocols, etc.

R2b **Flexible technology independent message structure:**
All messages are defined in ASN.1 enabling a self-descriptive human readable representation of the messages and at the same time efficient verifiable representation for machines.

R2c **Common encoding and decoding schema for messages:**
The use of ASN.1 allows to use multiple encoding rules to encode and decode the data. The ASN.1 PER (4.13) offers a very efficient encoding in terms of compression of the resulting byte stream, which is important for resource constrained devices.

R2d **Technology independent interaction protocols:**
All interactions in A3ME are based on the performatives, which are types of messages with a defined purpose. These performative based interaction protocols can be used independent of the technologies realizing those.

R2e **Technology independent query language:**
The query language developed in this work allows to exchange information independent of the technologies used on the individual devices.

R3 **Decentralized solution (DCS):**
The framework is designed to operate without central instances. It allows ad-hoc interactions between devices without external intervention.

R4 **Applicable to Heterogeneous Environments (Het):**
The flexibility of the framework allows to use it in various environments.

R4a **Applicable to WSN:**
As demonstrated in section 6.2.5 the framework can be used to collect sensor readings from different sensor networks in parallel.

R4b **Applicable to Ubiquitous Environments:**
For example A3ME enabled smartphones can be used to trigger commands to other devices.

R4c **Applicable to Unmanned Vehicles:**
As demonstrated with a prototype module in section 5.5 the framework can also be integrated into a robot operating system.

R5 **Low hardware requirements (LHW):**
In section 6.2 we showed in different experiments how the framework can be used on resource constrained devices like the TelosB platform.

7 Conclusions and Future Work

In this work we developed a framework starting with an abstract concept and evolved it step-by-step to a concrete system with corresponding prototypes (Figure 2). Hereby at each step we had to identify possible solutions and components to realize the next step, to evaluate the state of the art, and then select and develop an appropriate solution to realize the step.

During the development of this work we collected the requirements from different areas present in the heterogeneous environment of a MME. Nowadays many of these areas are growing together and have to interact in many ways. With respect to communication the wired telephony, mobile telephony, internet service providers and the television are merging together. Social networks, geo-mapping services and self-localization capabilities of the devices offer many new possibilities in all areas of our life. The devices grow in numbers and in their variety: sensors and actuators in the environment (smart home, smart office, connected car, smart city), devices carried by people (smart phones, computers, wearables), industrial automation, smart metering, etc. This increasing heterogeneity makes it obvious that we need a framework which simplifies the interactions among all these devices. The A3ME is such a framework.

While developing A3ME time we analyzed several middleware solutions, protocols and standards in different research areas. While offering specialized and convenient solutions for some areas, most of them are not generic enough to deal with heterogeneity as described before.

In this dissertation we designed the Device-Agent based Middleware for Mixed Mode Environments (A3ME) to cover all identified requirements and to apply different existing standards. What makes this work different from many other similar projects is that this work does not pretend to replace other specialized solutions, but reuses the existing specialized solutions and interconnects them with other solutions which exist in parallel.

The big challenge of this work was to design a framework enabling interactions among heterogeneous devices while keeping it simple and applicable to the different areas.

As our future work we would like to build new applications on top of A3ME framework, which now can use and benefit from the specialized solutions from the different research areas. One such application would be a generic graphical user interface (SMARTUI). Such a user interface would enable a user to discover and query sensor-measurements from nearby sensors, discover devices in a smart home and interact with them, discover a robot and control it, discover and allow interactions with other users via their devices. The information about discovered devices, capabilities, users etc. can be completed and looked up via semantic web mechanisms and visualized on a map. The collected information can then be used for further more complex tasks.

The prototypical implementations for the different devices can be optimized in different ways: identify and implement more efficient routing and discovery algorithms for the wireless sensor nodes, for the Bluetooth implementations the newer version 4.2 should be used together with the Bluetooth Low Energy (BLE) version of the stack. A light-weighted publish/subscribe service like MQTT could be set up on top of A3ME to optimize the interactions of the devices.

To extend the amount of reachable devices we will implement interfaces to other existing solutions in the areas: smart home, smart office, connected car, entertainment devices, smart metering, industrial automation, smart cities, social networks. The interconnection of these areas will offer many new possibilities and models for people's comfort, business, optimization and ecology.

Security and privacy issues, which were not covered in this dissertation have to be incorporated into the framework. Here different classes of required security and privacy for different classes of devices and application scenarios need to be identified. For these classes requirements must be defined. Then for each concrete set of devices and usage scenario the security and privacy classes need to be assigned, to be able to decide which sets of security technologies would satisfy the requirements. The identified set of technologies then needs to be applied/implemented to the participating devices. The same set of devices hereby can be used for different application scenarios. Depending on the application scenario the security and privacy requirements can be very different, e.g. the same set of sensors can be used for pet observation and for remote patient monitoring at home, where the second scenario has significantly higher security and privacy requirements.

This work can be used as the standard way for discovery and basic interactions of devices on smartphones and portable devices. The neutral representation of the devices and their capabilities and properties can be used as a neutral reference to interconnect different existing solutions and hereby enable and improve the overall interoperability of devices. The individual building blocks from this work can be reused in the evolving "smart" technologies: Smart Home, Connected Car, Internet of Things, Industry 4.0, Smart Cities, etc.

8 Glossary

A3ME	Device-Agent-based Middleware for Mixed Mode Environment (AMMME –> A3ME)	1
ASN.1	Abstract Syntax Notation One	3.8.5
CLDC	Connected Limited Device Configuration is a Java framework for building Java ME applications on limited embedded devices	5.2
CORBA	Common Object Request Broker Architecture	3.5.2
Device-agent	A program which runs on a device and represents it and its capabilities in a heterogeneous network.	4.2
EBNF	Extended Backus-Naur Form is a grammar definition language	3.5.2
FIPA	Foundation for Intelligent Physical Agents	3.5.6
IDL	Interface Definition Language	3.5.2
JAUS	Joint Architecture for Unmanned Systems is an international standard that defines communication protocols for unmanned vehicle systems	3.3.2
JNI	Java Native Interface	5.1
JSON	JavaScript Object Notation	3.8.4
KQML	Knowledge Query and Manipulation Language	3.11.2
MANETs	Mobile ad hoc networks	3.6.1
MIDP	Mobile Information Device Profile	5.2
Mobile agent	A program code together with its execution state and variables, which can move from one device to another and continue its execution there.	3.1.5
RMI	Remote Method Invocation	3.5.1
P2P	Peer-to-peer	3.5.4
ROS	Robot Operating System	5.5
RPC	Remote Procedure Call	3.5.1
Sensor ML	Sensor Markup Language	3.9.1
SMARTUI	Generic graphical user interface to control heterogeneous devices	7
SOAP	A protocol used to exchange structured information in a decentralized, distributed environment	3.8.3
TEDS	Transducer Electronic Data Sheet	3.9.2
WS	Web Service	3.5.3
WSDL	Web Services Description Language	3.5.3
WSN	Wireless Sensor Network	3.1
WWW	World Wide Web	3.5.3

References

[1] Information processing – Text and office systems – Standard Generalized Markup Language (SGML). Standard ISO 8879:1986, ISO, Aug. 1986.

[2] IEEE Standard for a Smart Transducer Interface for Sensors and Actuators-Network Capable Application Processor (NCAP) Information Model. Standard 1451.1-1999, IEEE, 1999.

[3] Jini(TM) Architecture Specification. Specification, Sun Microsystems, Inc., 1999.

[4] FIPA Interaction Protocol Library Specification. Standard (deprecated), Foundation for Intelligent Physical Agents, 2000.

[5] Abstract syntax notation one (ASN.1): Specification of basic notation. Recommendation x.680, ITU-T, 2002.

[6] ASN.1 Encoding Rules-Specification of Packed Encoding Rules (PER). Recommendation x.691, ITU-T, 2002.

[7] FIPA Abstract Architecture Specification. Standard, Foundation for Intelligent Physical Agents, 2002.

[8] FIPA ACL Message Representation in Bit-Efficient Encoding Specification. Standard, Foundation for Intelligent Physical Agents, 2002.

[9] FIPA ACL Message Structure Specification. Standard, Foundation for Intelligent Physical Agents, 2002.

[10] FIPA Communicative Act Library Specification. Standard, Foundation for Intelligent Physical Agents, 2002.

[11] Health Informatics – Point-Of-Care Medical Device Communication – Part 10101: Nomenclature. International Standard 11073-10101:2004(E), ISO/IEEE, 2004.

[12] Health informatics – Point-of-care medical device communication – Part 20101: Application profiles – Base standard. International Standard ISO-IEEE:11073-20101, ISO/IEEE, 2004.

[13] Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C recommendation, W3C, 2004.

[14] Crossbow TelosB Platform. Datasheet, Crossbow Technology, Inc., 2006.

[15] Wireless MAC and PHY Specifications for Low-Rate WPANs. Standard IEEE Std 802.15.4-2006, IEEE Computer Society, 2006.

[16] 2.4 GHz IEEE 802.15.4 / ZigBee-ready RF Transceiver. Documentation, Texas Instruments, 2007.

[17] IEEE Standard for a Smart Transducer Interface for Sensors and Actuators – Common Functions, Communication Protocols, and Transducer Electronic Data Sheet (TEDS) Formats. Standard 1451.0-2007, IEEE, 2007.

[18] JXTA v2.0 Protocols Specification. Specification, Sun Microsystems Inc., 2007.

[19] Common Object Request Broker Architecture (CORBA) for embedded. Specification Version 1.0, OMG Object Management Group, Inc., 2008.

[20] Extensible Markup Language (XML) 1.0 (Fifth Edition). Standard, W3C, 2008.

[21] Efficient XML Interchange Evaluation. Working draft, W3C Efficient XML Interchange (EXI) Working Group, Apr. 2009.

[22] Sun™ SPOT Theory of Operation. Documentation Release 5 (red), Sun Labs, 2009.

[23] MQTT V3.1 Protocol Specification. Specification, IBM, 2010.

[24] Sun™ SPOT Programmer's Manual. Documentation Release v6.0 (Yellow), Sun Labs, 2010.

[25] Z1 Datasheet. Datasheet v1.1, Zolertia, Mar. 2010.

[26] Common Object Request Broker Architecture (CORBA) - Part 1: CORBA Interfaces. Specification, OMG Object Management Group, Inc., 2011.

[27] Common Object Request Broker Architecture (CORBA) - Part 2: CORBA Interoperability. Specification, OMG Object Management Group, Inc., 2011.

[28] Common Object Request Broker Architecture (CORBA) - Part 3: CORBA Component Model. Specification, OMG Object Management Group, Inc., 2011.

[29] OGC© Sensor Planning Service Implementation Standard. Implementation standard, Open Geospatial Consortium, 2011.

[30] OGC© SWE Common Data Model Encoding Standard. Encoding Standard Version: 2.0.0, Open Geospatial Consortium Inc., 2011.

[31] OpenGIS© SWE Service Model. Implementation Standard Version: 2.0, Open Geospatial Consortium, 2011.

[32] Contiki Operating System Documentation. Documentation Version 2.5, 2012.

[33] OWL 2 Web Ontology Language Document Overview (Second Edition). Recommendation, W3C OWL Working Group, 2012.

[34] Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. IEEE Standard 802.11-2012, IEEE Computer Society, 2012.

[35] SPARQL 1.1 Overview. W3C recommendation, W3C SPARQL Working Group, 2013.

[36] SPARQL 1.1 Query Language. W3C recommendation, W3C SPARQL Working Group, 2013.

[37] Specification of the Bluetooth System. Specification 4.1, Bluetooth SIG, 2013.

[38] The JSON Data Interchange Format. Standard ECMA-404, ECMA International, 2013.

[39] Efficient XML Interchange (EXI) Format 1.0 (Second Edition). Recommendation, W3C, 2014.

[40] JSON - JavaScript Object Notation. Web page [accessed on 2014-01-29]. http://www.json.org, 2014.

[41] RDF Schema 1.1. W3C recommendation, W3C, 2014.

[42] E. Aitenbichler. System Support for Ubiquitous Computing. In *Advances in Pervasive Computing (submission to the Doctoral Colloquium at Pervasive 2004)*, pages 1–6. Austrian Computer Society (OCG), Austrian Computer Society (OCG), 2004.

[43] E. Aitenbichler. *System Support for Ubiquitous Computing*. Dissertation, Technische Universität Darmstadt, Aachen, Jan. 2006.

[44] I. F. Akyildiz and I. H. Kasimoglu. A protocol suite for wireless sensor and actor networks. In *Radio and Wireless Conference*, pages 11–14. IEEE, 2004.

[45] J. Al-Muhtadi, S. Chetan, A. Ranganathan, and R. Campbell. Super spaces: a middleware for large-scale pervasive computing environments. In *IEEE International Workshop on Pervasive Computing and Communications*, pages 198–203. IEEE, 2004.

[46] AllSeen Alliance. Overview of the AllSeen Alliance [accessed on 2015-07-11]. https://allseenalliance.org/sites/default/files/resources/intro_to_alliance_5.21.15.pdf, 2015.

[47] K. Arnold, R. Scheifler, J. Waldo, B. O'Sullivan, and A. Wollrath. *The Jini Specification*. Addison-Wesley, Boston, MA, USA, 1st edition, June 1999.

[48] M. Bakht, M. Trower, and R. Kravets. Searchlight: helping mobile devices find their neighbors. In *3rd ACM SOSP Workshop on Networking, Systems, and Applications on Mobile Handhelds*, page 9. ACM, 2011.

[49] O. Ben-Kiki and C. Evans. YAML Ain't Markup Language (YAML). Specification Version 1.2, 2009.

[50] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, and D. Orchard. Web Services Architecture. Technical report, Web Services Architecture W3C Working Group, 2004.

[51] M. Botts and A. Robin. OpenGIS Sensor Model Language (SensorML). Implementation Specification Version: 1.0.0, Open Geospatial Consortium Inc., 2007.

[52] A. Bröring, C. Stasch, and J. Echterhoff. Sensor Observation Service Interface Standard. Standard OGC 12-006, Open Geospatial Consortium, 2012.

[53] D. Bryan, V. Draluk, C. Kurt, J. Lancelle, S. Macroibeaird, A. T. Manes, B. Mckee, D. Ho, and J. Rodriguez. UDDI API Specification. Specification Version 2.04, OASIS, 2002.

[54] S. Chakrabarti, Z. Shelby, and E. Nordmark. Neighbor Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs). Request for Comments 6775, Internet Engineering Task Force (IETF), 2012.

[55] S. Chakrabarti, M. Wasserman, P. Thubert, and E. Nordmark. IPv6 Neighbor Discovery Optimizations for Wired and Wireless Networks. Internet-Draft Draft-05, Updates: RFC4861 (if approved), Internet Engineering Task Force (IETF), 2014.

[56] S. Chaudhuri and D. Mandal. Quorum System. Lecture notes Advanced topics on Networks and Distributed Algorithms, Iowa State University, Computer Science, 2014.

[57] H. Chen, T. Finin, and A. Joshi. An intelligent broker for context-aware systems. In *Adjunct proceedings of Ubicomp*, volume 3, pages 183–184, 2003.

[58] S. Cheshire and M. Krochmal. Multicast DNS. Request for comments rfc6762, Internet Engineering Task Force (IETF), Feb. 2013.

[59] T. Clausen, C. Dearlove, and J. Dean. Mobile Ad Hoc Network (MANET) Neighborhood Discovery Protocol (NHDP). Request for Comments RFC6130, Internet Engineering Task Force (IETF), 2011.

[60] M. Compton, P. Barnaghi, L. Bermudez, R. Garcia-Castro, O. Corcho, S. Cox, J. Graybeal, M. Hauswirth, C. Henson, A. Herzog, and Others. The SSN Ontology of the W3C Semantic Sensor Network Incubator Group. *Web Semantics: Science, Services and Agents on the World Wide Web*, 2012.

[61] S. Cox. Geographic information – Observations and measurements. Specification 19156:2011(E), OGC and ISO, 2011.

[62] C. Curino, M. Giani, M. Giorgetta, A. Giusti, A. L. Murphy, G. P. Picco, and P. Milano. TinyLIME: Bridging Mobile and Sensor Networks through Middleware. In *PERCOM '05: Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications*, pages 61–72, Washington, DC, USA, 2005. IEEE Computer Society.

[63] L. De Nardis, M.-G. Di Benedetto, V. Rakovic, V. Atanasovski, L. Gavrilovska, O. Holland, A. Akhtar, H. Aghvami, D. Tassetto, S. Bovelli, and Others. Neighbour and network discovery in cognitive radio networks: research activities and results in the ACROPOLIS Network of Excellence. *European Wireless 2013*, 2013.

[64] T. Dierks and C. Allen. RFC 2246: The TLS Protocol. Standard RFC 2246, IETF, 1999.

[65] A. Donoho, J. Costa-requena, T. Mcgee, A. Messer, A. Fiddian-green, and J. Fuller. UPnP™Device Architecture 1.1. Specification October, UPnP Forum, 2008.

[66] A. Donoho, B. Roe, M. Bodlaender, J. Gildred, A. Messer, Y. Kim, B. Fairman, and J. Tourzan. UPnP Device Architecture 2.0. Specification, UPnP Forum, Feb. 2015.

[67] O. A. Dragoi. *The Continuum Architecture: Towards Enabling Chaotic Ubiquitous Computing*. PhD thesis, University of Waterloo, 2004.

[68] O. A. Dragoi and J. P. Black. Enabling chaotic ubiquitous computing. Technical report, Technical Report CS-2004-35, University of Waterloo, Canada, 2004.

[69] O. Dubuisson. *ASN.1 Communication between Heterogeneous Systems*. OSS Nokalva, 2000.

[70] K. Ducatel, M. Bogdanowicz, F. Scapolo, J. Leijten, and J.-C. Burgelman. Scenarios for Ambient Intelligence in 2010. Technical report, European Commission, Joint Research Centre, Institute for Prospective Technological Studies (IPTS), Seville, 2001.

[71] P. Dutta and D. Culler. Practical asynchronous neighbor discovery and rendezvous for mobile sensing applications. In *Proceedings of the 6th ACM conference on Embedded network sensor systems*, SenSys '08, pages 71–84. ACM, ACM, 2008.

[72] W. K. Edwards, M. W. Newman, J. Sedivy, Trevor Smith, and S. Izadi. Challenge: Recombinant Computing and the Speakeasy Approach Categories and Subject Descriptors. In *Proceedings of the 8th annual international conference on Mobile computing and networking - MobiCom '02*, page 279, New York, New York, USA, Sept. 2002. ACM Press.

[73] F. Dawson and T. Howes. vCard MIME Directory Profile. RFC 2426, The Internet Engineering Task Force (IETF), 1998.

[74] T. Finin, J. Weber, G. Wiederhold, M. M. Genesereth, R. Fritzson, D. McKay, S. Shapiro, J. J. Mcguire, R. Pelavin, and C. Beck. Specification of the KQML Agent-Communication Language. Specification draft, The DARPA Knowledge Sharing Initiative External Interfaces Working Group, 1993.

[75] J. Fleischer, R. Häner, S. Herrnkind, A. Kloth, U. Kriegel, H. Schwarting, and J. Wächter. An integration platform for heterogeneous sensor systems in GITEWS – Tsunami Service Bus. *Natural Hazards and Earth System Science*, 10(6):1239–1252, 2010.

[76] A. C.-l. Fok, G.-C. Roman, C. Lu, and C.-L. Fok. Agilla: A Mobile Agent Middleware for Sensor Networks. Technical Report WUCSE-2006-16, Washington University, St. Louis, 2006.

[77] V. Galluzzi and T. Herman. Survey: Discovery in Wireless Sensor Networks. *International Journal of Distributed Sensor Networks*, pages 1–12, 2012.

[78] A. Gangemi, N. Guarino, C. Masolo, A. Oltramari, and L. Schneider. Sweetening ontologies with DOLCE. In *Knowledge engineering and knowledge management: Ontologies and the semantic Web*, pages 166–181. Springer, 2002.

[79] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler. The nesC language: A Holistic Approach to Networked Embedded Systems. *ACM SIGPLAN Notices*, 38(5):1, May 2003.

[80] T. Gu, H. K. Pung, and D. Q. Zhang. A middleware for building context-aware mobile services. In *59th Vehicular Technology Conference (VTC)*, volume 5, pages 2656–2660. IEEE, 2004.

[81] M. Gudgin, M. Hadley, N. Mendelsohn, J.-J. Moreau, Henrik Frystyk Nielsen, A. Karmarkar, and Y. Lafon. SOAP Version 1.2 Part 1: Messaging Framework (Second Edition). W3C Recommendation 27 April 2007, W3C, 2007.

[82] M. Gudgin, M. Hadley, N. Mendelsohn, J.-J. Moreau, H. F. Nielsen, A. Karmarkar, and Y. Lafon. SOAP Version 1.2 Part 2: Adjuncts (Second Edition). W3C Recommendation 27 April 2007, W3C, 2007.

[83] P. E. Guerrero, A. P. Buchmann, A. Khelil, and K. Van Laerhoven. TUDμNet, a Metropolitan-Scale Federation of Wireless Sensor Network Testbeds. In *9th European Conference on Wireless Sensor Networks*, Feb. 2012.

[84] P. E. Guerrero, I. Gurov, S. Santini, and A. Buchmann. On the Selection of Testbeds for the Evaluation of Sensor Network Protocols and Applications. In *14th IEEE Workshop on Signal Processing Advances in Wireless Communications*, SPAWC 2013, pages 490–494. IEEE, 2013.

[85] P. E. Guerrero, D. Jacobi, and A. Buchmann. Workflow Support for Wireless Sensor and Actor Networks A Position Paper. In *4th International Workshop on Data Management for Sensor Networks*, Vienna, Austria, 2007.

[86] S. Hadim and N. Mohamed. Middleware: Middleware Challenges and Approaches for Wireless Sensor Networks. In *IEEE distributed systems online*, volume 7. IEEE Computer Society, 2006.

[87] S. Y. Han and D. Lee. An Adaptive Hello Messaging Scheme for Neighbor Discovery in On-Demand MANET Routing Protocols. *IEEE Communications Letters*, 17(5):1040–1043, May 2013.

[88] K. Henricksen and R. Robinson. A Survey of Middleware for Sensor Networks: State-of-the-Art and Future Directions. In *International Workshop on Middleware for sensor networks - MidSens '06*, pages 60–65, New York, New York, USA, Nov. 2006. ACM Press.

[89] Herqq.org. Tutorial for Building a UPnP Device [online accessed on 2012-05-21]. http://www.herqq.org/html/herqq_trunk/builddevice_tutorial.html, 2011.

[90] A. Herzog and A. Buchmann. Predefined Classification for Mixed Mode Environments. Technical report, Technische Universität Darmstadt, Darmstadt, Aug. 2009.

[91] A. Herzog and A. Buchmann. A3ME - Generic Middleware for Heterogeneous Environments. In *International Conference on Networked Sensing Systems (INSS'12)*, 2012.

[92] A. Herzog, D. Jacobi, and A. Buchmann. A3ME - An Agent-Based Middleware Approach for Mixed Mode Environments. In *International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM)*, pages 191–196. IEEE, Sept. 2008.

[93] I. Hickson. HTML Microdata. Specification draft, W3C, 2012.

[94] B. Horling, R. Mailler, and V. Lesser. A Case Study of Organizational Effects in a Distributed Sensor Network. In *ACM International Conference on Intelligent Agent Technology*. IEEE, 2004.

[95] C. Hoss and M. Weyland. *openASN.1 : Entwicklung und Evaluation eines ASN.1-Compilers und PER-Codecs unter Java*. Diplomarbeit, Technische Universität Darmstadt, 2007.

[96] J. Hyyrylaäinen and I. Jantunen. SSI protocol specification. Specification Version 1.2, Nokia Research Center, 2006.

[97] ITU Telecommunication Standardization Sector (ITU-T). Application fields of ASN.1. [online accessed on 2014-01-12]. http://www.itu.int/ITU-T/asn1/uses/.

[98] D. Jacobi, P. E. Guerrero, K. Nawaz, C. Seeger, A. Herzog, K. V. Laerhoven, and I. Petrov. Towards Declarative Query Scoping in Sensor Networks. In K. Sachs, I. Petrov, and P. Guerrero, editors, *From Active Data Management to Event-Based Systems and More*, Lecture Notes in Computer Science, chapter Towards De, pages 281–292. Springer, 2010.

[99] L. Kagal and T. Finin. A policy language for a pervasive computing environment. In *4th International Workshop on Policies for Distributed Systems and Networks*, pages 63–74. IEEE Computer Society, 2003.

[100] A. Kandhalu, K. Lakshmanan, and R. R. Rajkumar. U-connect: A Low-latency Energy-efficient Asynchronous Neighbor Discovery Protocol. In *9th ACM/IEEE International Conference on Information Processing in Sensor Networks*, IPSN '10, pages 350–361, New York, NY, USA, 2010. ACM.

[101] S. Khatibi and R. Rohani. Quorum-based neighbor discovery in self-organized cognitive MANET. In *Personal Indoor and Mobile Radio Communications (PIMRC), 2010 IEEE 21st International Symposium on*, pages 2239–2243, Sept. 2010.

[102] N. Koshizuka and K. Sakamura. Ubiquitous ID: Standards for Ubiquitous Computing and the Internet of Things. *Pervasive Computing, IEEE*, 9(4):98–101, 2010.

[103] K. Kreuzer. Openhab – Open Home Automation Bus [accessed on 2014-03-31]. http://openhab.org.

[104] M. Kropff. Sensorbasiertes Monitoring zur kontextsensitiven Unterstützung von Wissensarbeit. Dissertation, Technische Universität Darmstadt, Fachbereich Elektrotechnik und Informationstechnik, Darmstadt, Sept. 2011.

[105] Y. Labrou and T. Finin. A Proposal for a new KQML Specification. Specification, University of Maryland Baltimore County (UMBC), 1997.

[106] S. Lai. Heterogenous Quorum-based Wakeup Scheduling for Duty-Cycled Wireless Sensor Networks Heterogenous Quorum-based Wakeup Scheduling for Duty-Cycled. Dissertation, Virginia Polytechnic Institute and State University, 2009.

[107] A. Lappeteläinen, M. H. N. Vääräkangas, H. Laine, D. Trossen, and D. Pavel. Overall MIMOSA architecture specification (OMAS). Deliverable report D2.1[2], European Project FP6, 2005.

[108] L. Lefort, C. Henson, K. Taylor, P. Barnaghi, M. Compton, O. Corcho, R. G. Castro, J. Graybeal, A. Herzog, K. Janowicz, H. Neuhaus, A. Nikolov, and K. Page. Semantic Sensor Network XG Final Report. Final report, W3C Semantic Sensor Network Incubator Group (SSN-XG), 2011.

[109] P. Levis and D. E. Culler. Maté: A Tiny Virtual Machine for Sensor Networks. In *10th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 85–95, New York, NY, USA, 2002. ACM Press.

[110] P. Levis and D. Gay. *TinyOS Programming*. Cambridge University Press, Cambridge, 2009.

[111] J. Li, J. Yang, Y. Zhang, L. Guo, and Y. Li. Neighbor Discovery Algorithm Based on the Regulation of Duty-cycle in Mobile Sensor Network. In *Proceedings of the 8th International Conference on Wireless Algorithms, Systems, and Applications*, WASA'13, pages 285–299, Berlin, Heidelberg, 2013. Springer-Verlag.

[112] J. Liu, C. Chen, Y. Ma, and Y. Xu. Adaptive Device Discovery in Bluetooth Low Energy Networks. In *77th Vehicular Technology Conference (VTC Spring)*, pages 1–5, 2013.

[113] J. Liu and V. Issarny. QoS-aware Service Location in Mobile Ad-Hoc Networks. In *5th IEEE International Conference on Mobile Data Management*, pages 224–235, 2004.

[114] V. Lortz and M. Saaranen. UPnP DeviceProtection:1 Service. Standardized Device Control Protocol Version 1.0, UPnP Forum, 2011.

[115] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TinyDB: an acquisitional query processing system for sensor networks. *ACM Transactions on Database Systems*, 30(1):122–173, 2005.

[116] S. Maffeis. Communication Middleware for Mobile Applications – A Comparison. Technical report, Softwired AG, 2001.

[117] M. J. McGlynn and S. A. Borbash. Birthday protocols for low energy deployment and flexible neighbor discovery in ad hoc wireless networks. In *Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking & computing*, pages 137–145. ACM, 2001.

[118] A. Messer. UPnP Middleware Developers Event Face-to-Face Presentation. http://upnp.org/resources/documents/UPnPForum_MiddlewareDevEvent_March2012.pdf, 2012.

[119] Microformats.org. Microformats description. Web page [accessed on 2014-04-06]. http://microformats.org/about.

[120] V. Misra. Oracle®Fusion Middleware Reference for Oracle Security Developer Tools. Documentation February, Oracle, 2013.

[121] N. Mitra and Y. Lafon. SOAP Version 1.2 Part 0: Primer (Second Edition). W3C Recommendation 27 April 2007, W3C, 2007.

[122] S. B. Mokhtar, D. Preuveneers, N. Georgantas, V. Issarny, and Y. Berbers. EASY: Efficient semAntic Service discoverY in pervasive computing environments with QoS and context support. *Journal of Systems and Software*, 81(5):785–808, 2008.

[123] Moteiv Corporation. TelosB Platform. Technical report, Moteiv Corporation, 2004.

[124] M. Mühlhäuser, E. Aitenbichler, G. Austaller, A. Hartl, A. Heinemann, and C. Trompler. Towards Personalized Ubiquitous Computing Services. Technical report, Fachbereich Informatik, Technische Universität Darmstadt, Aug. 2002.

[125] K. Nahrstedt, D. Xu, D. Wichadakul, and B. Li. QoS-aware middleware for ubiquitous and heterogeneous environments. *IEEE Communications Magazine*, 39(11):140–148, 2001.

[126] T. Narten, E. Nordmark, W. Simpson, and H. Soliman. Neighbor Discovery for IP version 6 (IPv6). Request for Comments RFC4861, Internet Engineering Task Force (IETF), Sept. 2007.

[127] E. Niemelä and J. Latvakoski. Survey of requirements and solutions for ubiquitous software. In *Proceedings of the 3rd international conference on Mobile and ubiquitous multimedia - MUM '04*, pages 71–78, New York, New York, USA, Oct. 2004. ACM Press.

[128] Open Source Robotics Foundation. ROS Robot Operating System web page [accessed on 2014-03-23]. http://www.ros.org.

[129] Oracle. Java Naming and Directory Interface (JNDI). http://www.oracle.com/technetwork/java/jndi/, 2014.

[130] OSS Nokalva. ASN.1 Encoding Rules Description [online accessed on 2013-12-17]. http://www.oss.com/asn1/rules.html.

[131] H. Pennington, A. Carlsson, A. Larsson, S. Herzberg, S. McVittie, and D. Zeuthen. D-Bus Specification. Specification Revision 0.26, freedesktop.org, 2015.

[132] J. Polastre, J. Hill, and D. Culler. Versatile low power media access for wireless sensor networks. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 95–107. ACM, 2004.

[133] D. Preuveneers and Y. Berbers. Semantic and syntactic modeling of component-based services for context-aware pervasive systems using owl-s. *First International Workshop on Managing Context Information in Mobile and Pervasive Environments*, pages 30–39, 2005.

[134] D. Preuveneers and Y. Berbers. Towards energy-aware semantic publish/subscribe for wireless embedded systems. *ICST Transactions on Ubiquitous Environments*, 12(10-12), 2012.

[135] D. Preuveneers, J. den Bergh, D. Wagelaar, A. Georges, P. Rigole, T. Clerckx, Y. Berbers, K. Coninx, V. Jonckers, and K. De Bosschere. Towards an extensible context ontology for ambient intelligence. In *Ambient intelligence*, pages 148–159. Springer Berlin Heidelberg, 2004.

[136] Project Kenai. JXTA The Language and Platform Independent Protocol for P2P Networking. Web page [accessed on 2011-10-18]. http://jxta.kenai.com/.

[137] C. Reed, M. Botts, G. Percivall, and J. Davidson. OGC® Sensor Web Enablement: Overview And High Level Architecture. White Paper OGC 07-165r1, Open Geospatial Consortium, Apr. 2013.

[138] M. Román, C. Hess, R. Cerqueira, R. Campbell, and K. Nahrstedt. Gaia: A middleware infrastructure to enable active spaces. *IEEE Pervasive Computing*, 1(4):74–83, 2002.

[139] K. Römer, O. Kasten, and F. Mattern. Middleware Challenges for Wireless Sensor Networks. *Mobile Computing and Communications Review*, 6(4):59–61, Oct. 2002.

[140] SAE International. OpenJAUS - JAUS Robotics Software Development Kit (SDK) [online accessed on 2011-10-11]. http://www.openjaus.com.

[141] Samsung.com. Samsung Nexus S Datasheet [online accessed on 2014-04-04]. http://www.samsung.com/de/support/model/GT-I9023FSADBT-techspecs, 2014.

[142] J. Schmitt. *Anpassungsfähige Kontextbestimmung zur Unterstützung von Kommunikationsdiensten*. Dissertation, Technische Universität Darmstadt, Fachbereich Elektrotechnik und Informationstechnik, Multimedia Kommunikation, 2009.

[143] J. Schmitt, M. Kropff, A. Reinhardt, and M. Hollick. ContextFramework.KOM – Eine offene Middleware zur Integration heterogener Sensoren in eine Kontext-sensitive Kommunikationsplattform. Software Demonstration, KuVS Software Preis, Feb. 2009.

[144] Z. Shelby, K. Hartke, and C. Bormann. Constrained Application Protocol (CoAP). Proposed Standard draft-ietf-core-coap-18, Internet Engineering Task Force (IETF), 2013.

[145] A. Sheth, C. Henson, and S. S. Sahoo. Semantic sensor web. *Internet Computing, IEEE*, 12(4):78–83, 2008.

[146] A. Stanford-Clark and H. L. Truong. MQTT For Sensor Networks (MQTT-SN). Protocol Specification Version 1.2, IBM, 2013.

[147] T. Strang, C. Linnhoff-Popien, and K. Frank. CoOL: A context ontology language to enable contextual interoperability. In *Distributed applications and interoperable systems*, pages 236–247. Springer, 2003.

[148] R. Tan, J. Gu, Z. Zhong, and P. Chen. SOCOM: Multi-sensor Oriented Context Model Based on Ontologies. In *Eighth International Conference on Intelligent Environments*, pages 236–242. IEEE, June 2012.

[149] D. I. Tapia, R. S. Alonso, F. De la Prieta, C. Zato, S. Rodriguez, E. Corchado, J. Bajo, and J. M. Corchado. SYLPH: An Ambient Intelligence based platform for integrating heterogeneous Wireless Sensor Networks. In *International Conference on Fuzzy Systems (FUZZ)*, pages 1–8. IEEE, July 2010.

[150] Telecom Italia. JADE Architecture Overview. [online accessed on 2012-05-12]. http://jade.tilab.com/doc/tutorials/JADEAdmin/jadeArchitecture.html, 2012.

[151] The Apache Software Foundation. Apache River - User Guide [accessed on 2012-05-09]. http://river.apache.org/user-guide-basic-river-services.html.

[152] A. Toninelli, A. Corradi, and R. Montanari. Semantic-based discovery to support mobile context-aware service access. *Computer Communications*, 31(5):935–949, 2008.

[153] Y.-C. Tseng, C.-S. Hsu, and T.-Y. Hsieh. Power-saving protocols for IEEE 802.11-based multi-hop ad hoc networks. *Computer Networks*, 43(3):317–337, 2003.

[154] V. Venturini, J. Carbó, and J. M. Molina. An Ambient Intelligent Platform based on Multi-Agent System. In *11 th Workshop of Physical Agents*, 2010.

[155] WAP Forum. Wireless Application Protocol – User Agent Profile (UAProf). Specification Version 20 October 2001, Wireless Application Forum, Ltd, 2001.

[156] M. Weiser and J. S. Brown. The coming age of calm technology. In *Beyond calculation*, pages 75–85. Springer, 1997.

[157] J. Weppner and P. Lukowicz. Collaborative Crowd Density Estimation with Mobile Phones. In *9th ACM Conf. Embedded Networked Sensor Systems*, 2011.

[158] C. Willcock. A Tutorial Introduction to ASN.1 97. *Telektronikk*, 4:62–69, 2000.

[159] Y. Yao and J. Gehrke. The cougar approach to in-network query processing in sensor networks. *SIGMOD record*, 31(3), 2002.

[160] T. Yashiro, S. Kobayashi, N. Koshizuka, and K. Sakamura. An Internet of Things (IoT) architecture for embedded appliances. In *Humanitarian Technology Conference (R10-HTC), 2013 IEEE Region 10*, pages 314–319, Aug. 2013.

[161] M. Zhang, L. Zhang, P. Yang, and Y. Yan. McDisc: A Reliable Neighbor Discovery Protocol in Low Duty Cycle and Multi-channel Wireless Networks. In *Networking, Architecture and Storage (NAS), 2013 IEEE Eighth International Conference on*, pages 1–7, 2013.

A Appendix

A.1 Classification List with Numeric Encodings

	Level 1	Level 2	Level 3	Code
A3ME				0
	ID			1
		Local ID		2
		Global ID		3
		Other ID		4
	Device			5
		Tag		6
		Mote		7
		Mobile		8
		Workstation		9
		Server		10
		Vehicle		11
		Multimedia		12
		Set of Devices		13
		Other Device		14
	Capability			15
		Sensor		16
			Switch	17
			Temperature	18
			Light	19
			Humidity	20
			Acceleration	21
			Voltage	22
			Position	23
			Distance	24
			Sound	25
			Vision	26
			Vibration	27
			Radiation	28
			Chemical	29
			Other Sensor	30
		Actuator		31
			Switch Controller	32
			Device Controller	33
			Motion	34
			Manipulator	35
			Other Actuator	36
		HID		37
			Input	38
			Output	39
			InOut	40
			Other HID	41
		Energy		42
			Not Limited	43
			Battery	44
			Renewable	45
			Passive	46
			Other Energy	47
		Storage		48
			ROM	49

Continued on next page

Table 19 – continued from previous page

	Level 1	Level 2	Level 3	Code
			RAM	50
			Flash	51
			HD	52
			Other Storage	53
		Communication		54
		Computing		55
		Other Capability		56
	Service			57
		Hardware Service		58
		Software Service		59
		Real World Service		60
		Other Service		61
	Data			62
		Number		63
		Text		64
		Date		65
		Record		66
		Array		67
		Stream		68
		Other Data		69
	Property			70
	Other			71

Table 19: A3ME Classification Codes.

A.2 ASN.1 Definition of the A3ME Classification

```
/**
 * Author: Arthur Herzog
 * Created: 2010-12-20 14:20:52 CET
 * Updated: 2011-09-15
 */

A3meOntology DEFINITIONS AUTOMATIC TAGS ::= BEGIN

-- imports and exports

EXPORTS A3ME-code;
IMPORTS ;

A3ME-code ::= ENUMERATED {
  a3me     (0),
  id       (1),
  local-id  (2),
  global-id (3),
  other-id  (4),
  device   (5),
  tag      (6),
  mote     (7),
  mobile   (8),
  workstation (9),
  server   (10),
  vehicle  (11),
  multimedia  (12),
  set-of-devices  (13),
  other-device  (14),
  capability  (15),
  sensor   (16),
  switch   (17),
  temperature (18),
  light    (19),
```

```
    humidity  (20),
    acceleration  (21),
    voltage (22),
    position  (23),
    distance  (24),
    sound  (25),
    vision  (26),
    vibration (27),
    radiation (28),
    chemical  (29),
    other-sensor  (30),
    actuator  (31),
    switch-controller (32),
    device-controller (33),
    motion  (34),
    manipulator (35),
    other-actuator  (36),
    hid  (37),
    input  (38),
    output  (39),
    inout  (40),
    other-hid (41),
    energy  (42),
    not-limited (43),
    battery (44),
    renewable (45),
    passive (46),
    other-energy  (47),
    storage (48),
    rom  (49),
    ram  (50),
    flash  (51),
    hd  (52),
    other-storage (53),
    communication (54),
    computing  (55),
    other-capability  (56),
    service (57),
    hardware-service  (58),
    software-service  (59),
    real-world-service  (60),
    other-service (61),
    data  (62),
    number  (63),
    text  (64),
    date  (65),
    record  (66),
    array  (67),
    stream  (68),
    other-data  (69),
    property  (70),
    other  (71),
    ... -- further possible extensions
}

END
```

Listing 43: a3meOntology.asn

A.3 ASN.1 Definition of the A3ME Message Parameters

```
/**
 * Author: Arthur Herzog
 * Created: 2011-02-10 11:17:40 CET
 * Updated: 2011-09-13
 */
A3meMessage DEFINITIONS AUTOMATIC TAGS ::= BEGIN

-- imports and exports
```

```
EXPORTS Message;
IMPORTS
  Message-content, Address, Addresses, DaID, StringType, GeneralizedTimeString
  FROM A3meContent;

-- type assignments
Message ::= SEQUENCE {
    performative  Performative,
    sender        DaID      OPTIONAL, --Denotes the identity of the sender of the message
    receiver      Addresses OPTIONAL,
    reply-to      DaID      OPTIONAL,
    content Message-content OPTIONAL,
    language  Language  OPTIONAL,
    encoding      Encoding  OPTIONAL,
    ontology      Ontology  OPTIONAL,
    protocol      Protocol  OPTIONAL,
    conversation-id ConversationID  OPTIONAL,
    reply-with  ConversationID  OPTIONAL,
    in-reply-to   ConversationID  OPTIONAL,
    reply-by      GeneralizedTime OPTIONAL,
    received-from Address         OPTIONAL --
}

Performative ::= ENUMERATED {
  /** FIPA performative constants **/
  accept-proposal ,--(0),
  agree           ,--(1),
  cancel          ,--(2),
  cfp             ,--(3),
  confirm         ,--(4),
  disconfirm      ,--(5),
  failure         ,--(6),
  inform          ,--(7),
  inform-if       ,--(8),
  inform-ref      ,--(9),
  not-understood  ,--(10),
  propose         ,--(11),
  query-if        ,--(12),
  query-ref       ,--(13),
  refuse          ,--(14),
  reject-proposal ,--(15),
  request         ,--(16),
  request-when    ,--(17),
  request-whenever,--(18),
  subscribe       ,--(19),
  proxy           ,--(20),
  propagate       ,--(21),
  unknown         --(-1) represented as 22 here
}

Language ::= ENUMERATED {
  a3me-language (0),
  ...
}

Encoding ::= ENUMERATED {
  asn-uper (0),
  ...
}

Ontology ::= ENUMERATED {
  a3me-ontolgy (0),
  ...
}

Protocol ::= ENUMERATED {
  request,
  query,
  request-when,
```

```
  contract-net,
  iterated-contract-net,
  brokering,
  recruiting,
  subscribe,
  propose,
  ...
}

ConversationID ::= SEQUENCE {
  owner    StringType OPTIONAL, -- DaID.name
  conversationNr  INTEGER(0..4294967295)
}

-- value assignments
END
```

Listing 44: a3meMessage.asn

A.4 ASN.1 Definition of the A3ME Content Data

```
/**
 * Author: Arthur Herzog
 * Created: Mon Dec 20 14:20:52 CET 2010
 */

A3meContent DEFINITIONS AUTOMATIC TAGS ::= BEGIN

-- imports and exports

EXPORTS Message-content, Address, Addresses, DaID, StringType, GeneralizedTimeString;
IMPORTS
  A3ME-code
  FROM A3meOntology;

Message-content ::= CHOICE {
  request-content      Request-content,
  inform-content       Inform-content,
  refuse-content       Refuse-content,
  not-understood-content  Not-understood-content,
  cancel-content       Cancel-content,
  encrypted-content    Encrypted-content,
  ...
}

Request-content ::= SEQUENCE {
  what  What,
  from  From-clause    OPTIONAL,
  where Condition-clause  OPTIONAL,
  period  Period-clause    OPTIONAL,
  range Range-clause     OPTIONAL
}

What ::= CHOICE {
  data-columns Data-descriptors,
  service-call Service-call
}

Data-descriptors  ::= SEQUENCE (SIZE(0..1023)) OF Data-descriptor

Data-descriptor   ::= SEQUENCE {
  a3me-code A3ME-code,
  infotype Infotype
}

Service-call ::= SEQUENCE {
  service CHOICE {
    id INTEGER(0..65535),
    capability-code A3ME-code
```

```
    --service-code A3ME-code
  },
  command INTEGER(0..1023),
  parameters Record OPTIONAL
}

Infotype ::= ENUMERATED {
  type-code,     --encoding number from the A3ME ontology.
  type-name,     --type name from the A3ME ontology.
  name,          --human readable Name of the object.
  description,   --human readable descr. of the object.
  id,            --ID for the object
  data,          --data value(s) (e.g. sensor readings)
  m2m-description,--machine readable descr (e.g. WSDL doc.)
  ...            --possible extensions
}

From-clause   ::= SEQUENCE (SIZE (0..1023)) OF DaID --[FROM (ALL / daID *[, daID]) ]

DaID       ::= SEQUENCE {
  name       StringType,
  addresses Addresses (SIZE(1..64)) OPTIONAL
}

Addresses   ::= SEQUENCE (SIZE (0..1023)) OF Address

Address     ::= SEQUENCE {
  address-type  StringType (SIZE(1..16)),
  address       StringType (SIZE(1..256))
}

Condition-clause  ::= SEQUENCE (SIZE (0..1023)) OF Condition --[WHERE condition *[AND condition]]

Condition   ::= CHOICE {
  a3me-code      A3ME-code,       --existence condition of given a3me-code on DA
  is-for-condition  Is-for-condition, -- IS-FOR data-descriptor a3me-code e.g. service for temp
  operator-condition  Operator-condition  -- operator data-descriptor string
}

Is-for-condition  ::= SEQUENCE {
  data-descriptor Data-descriptor,
  a3me-code A3ME-code
}

Operator-condition  ::= SEQUENCE {
  operator     Operator,
  data-descriptor Data-descriptor,
  parameter    Data-item
}

Operator  ::= ENUMERATED {
  equals,
  greater,
  greater-equal,
  smaller,
  smaller-equal,
  ...          --possible extensions
}

Period-clause ::= SEQUENCE {
  period Time-value,
  duration Time-value OPTIONAL
}

Time-value  ::= SEQUENCE {
  number INTEGER(0..4294967295),
  time-unit Time-unit
}

Time-unit ::= ENUMERATED {
```

```
    nanosecond,
    millisecond,
    second,
    minute,
    hour,
    day,
    week,
    year,
    ...         --possible extensions
}

Range-clause  ::= SEQUENCE {
  number INTEGER(0..4294967295),
  distance-unit Distance-unit
}

Distance-unit ::= ENUMERATED {
  hop,
  meter,
  kilometer,
  ...         --possible extensions
}

Inform-content  ::= SEQUENCE {
  sequence-number INTEGER(0..4294967295) OPTIONAL,
  resultset Resultset
}

Resultset ::= SEQUENCE {
  schema  Data-descriptors,
  rows SEQUENCE (SIZE (0..65535)) OF Data-record
}

Record     ::= SEQUENCE (SIZE (0..1023)) OF Data-item

-- Record containing only data to be used inside resultset
Data-record    ::= SEQUENCE (SIZE (0..1023)) OF Data

Data-item ::= SEQUENCE {
  data-descriptor Data-descriptor OPTIONAL,
  data Data OPTIONAL
}

Data ::= CHOICE {
  integer-data INTEGER(-2147483648..2147483647),
  real-data REALType, -- REAL type not supported by oss tool for JavaME
  boolean-data BOOLEAN,
  string-data StringType(SIZE(0..65000)),
  -- GeneralizedTime support is disabled due to encoder size constraints on small devices therefore useng
       a string version
  date-data    GeneralizedTimeString,
  time-data Time-data,
  byte-data OCTET STRING(SIZE (0..65000)),
  bit-string BIT STRING(SIZE (0..65000)),
  null NULL,
  record-data Record,
  --key-value-pair     Key-value-pair,
  ...
}

REALType       ::= SEQUENCE { /* Definition from ASN.1-X.680 */
  mantissa INTEGER(-2147483648..2147483647),
  base INTEGER (2|10),
  exponent INTEGER(-46340..46340)
  -- The associated mathematical real number is "mantissa"
  -- multiplied by "base" raised to the power "exponent"
}

Time-data ::= SEQUENCE(SIZE (0..10)) OF Time-value
```

```
Refuse-content  ::= SEQUENCE {    --REFUSE requestID [string]
  reason StringType(SIZE(0..1023)) OPTIONAL
}

Not-understood-content  ::= SEQUENCE {
  reason StringType(SIZE(0..1023)) OPTIONAL
}

Cancel-content  ::= SEQUENCE {
  reason StringType(SIZE(0..1023)) OPTIONAL
}

-- StringType ::= UTF8String(SIZE (0..65535))
StringType  ::= IA5String(SIZE (0..65535))

-- StringType ::= OCTET STRING (SIZE (0..65535))
-- (1) the octet string must contain a UTF-8 string and
-- (2) the numbers in the size constraint refer to the number of bytes and not to the number of
    characters

-- GeneralizedTime support is disabled due to encoder size constraints on small devices therefore useng a
    string version
GeneralizedTimeString ::= VisibleString (SIZE (18..24))

Encrypted-content ::= SEQUENCE {
  ciphertext    OCTET STRING(SIZE(0..65000)),
  algorithm   StringType(SIZE(0..127)) OPTIONAL,
  encrypted-for StringType(SIZE(0..127)) OPTIONAL
}

  -- value assignments
END
```

Listing 45: a3meContent.asn

A.5 ASN.1 Definition of the A3ME Object Identifiers

```
/**
 * Author: Arthur Herzog
 * Created: 2012-02-23
 */

A3meOntologyOIDs DEFINITIONS AUTOMATIC TAGS ::= BEGIN

-- imports and exports

EXPORTS A3ME-OID;
IMPORTS ;

--urn:oid:1.3.6.1.4.1.8301
copmuter-science-tu-darmstadt  OBJECT IDENTIFIER ::= {iso(1) identified-organization(3) dod(6) internet
    (1) private(4) enterprise(1) 8301}
dvs           OBJECT IDENTIFIER ::= {copmuter-science-tu-darmstadt dvs(4)}
a3me-oid-root OBJECT IDENTIFIER ::= {dvs projects(0)}

A3ME-OID  ::= SEQUENCE {a3me-oid-root RELATIVE-OID}

roid-a3me   RELATIVE-OID ::= {a3me(0)}    -- a3me constant (0),

roid-id   RELATIVE-OID ::= {roid-a3me id(1)}    -- a3me constant (1),

roid-local-id RELATIVE-OID ::= {roid-id local-id(1)} -- a3me constant (2),
roid-global-id  RELATIVE-OID ::= {roid-id global-id(2)} -- a3me constant (3),
roid-other-id RELATIVE-OID ::= {roid-id other-id(0)} -- a3me constant (4),

roid-device RELATIVE-OID ::= {roid-a3me device(2)} -- a3me constant (5),

roid-tag  RELATIVE-OID ::= {roid-device tag(1)} -- a3me constant (6),
roid-mote RELATIVE-OID ::= {roid-device mote(2)} -- a3me constant (7),
```

```
roid-mobile RELATIVE-OID ::= {roid-device mobile(3)}  -- a3me constant (8),
roid-workstation  RELATIVE-OID ::= {roid-device workstation(4)} -- a3me constant (9),
roid-server RELATIVE-OID ::= {roid-device server(5)}  -- a3me constant (10),
roid-vehicle  RELATIVE-OID ::= {roid-device vehicle(6)} -- a3me constant (11),
roid-multimedia RELATIVE-OID ::= {roid-device multimedia(7)}  -- a3me constant (12),
roid-set-of-devices RELATIVE-OID ::= {roid-device set-of-devices(8)}-- a3me constant (13),
roid-other-device RELATIVE-OID ::= {roid-device other-device(0)}  -- a3me constant (14),

roid-capability RELATIVE-OID ::= {roid-a3me capability(2)} -- a3me constant (15),

roid-sensor RELATIVE-OID ::= {roid-capability sensor(1)}  -- a3me constant (16),
roid-switch RELATIVE-OID ::= {roid-sensor switch(1)}  -- a3me constant (17),
roid-temperature  RELATIVE-OID ::= {roid-sensor temperature(2)}  -- a3me constant (18),
roid-light  RELATIVE-OID ::= {roid-sensor light(3)}  -- a3me constant (19),
roid-humidity RELATIVE-OID ::= {roid-sensor humidity(4)}  -- a3me constant (20),
roid-acceleration RELATIVE-OID ::= {roid-sensor acceleration(5)}  -- a3me constant (21),
roid-voltage  RELATIVE-OID ::= {roid-sensor voltage(6)}  -- a3me constant (22),
roid-position RELATIVE-OID ::= {roid-sensor position(7)}  -- a3me constant (23),
roid-distance RELATIVE-OID ::= {roid-sensor distance(8)}  -- a3me constant (24),
roid-sound  RELATIVE-OID ::= {roid-sensor sound(9)}  -- a3me constant (25),
roid-vision RELATIVE-OID ::= {roid-sensor vision(10)}  -- a3me constant (26),
roid-vibration  RELATIVE-OID ::= {roid-sensor vibration(11)}  -- a3me constant (27),
roid-radiation  RELATIVE-OID ::= {roid-sensor radiation(12)}  -- a3me constant (28),
roid-chemical RELATIVE-OID ::= {roid-sensor chemical(13)}  -- a3me constant (29),
roid-other-sensor RELATIVE-OID ::= {roid-sensor other-sensor(0)}  -- a3me constant (30),
roid-actuator RELATIVE-OID ::= {roid-capability actuator(2)}  -- a3me constant (31),
roid-switch-controller  RELATIVE-OID ::= {roid-actuator switch-controller(1)}  -- a3me constant (32),
roid-device-controller  RELATIVE-OID ::= {roid-actuator device-controller(2)}  -- a3me constant (33),
roid-motion RELATIVE-OID ::= {roid-actuator motion(3)}  -- a3me constant (34),
roid-manipulator  RELATIVE-OID ::= {roid-actuator manipulator(4)}  -- a3me constant (35),
roid-other-actuator RELATIVE-OID ::= {roid-actuator other-actuator(0)}  -- a3me constant (36),
roid-hid  RELATIVE-OID ::= {roid-capability hid(3)}  -- a3me constant (37),
roid-input  RELATIVE-OID ::= {roid-hid input(1)}  -- a3me constant (38),
roid-output RELATIVE-OID ::= {roid-hid output(2)}  -- a3me constant (39),
roid-inout  RELATIVE-OID ::= {roid-hid inout(3)}  -- a3me constant (40),
roid-other-hid  RELATIVE-OID ::= {roid-hid other-hid(0)}  -- a3me constant (41),
roid-energy RELATIVE-OID ::= {roid-capability energy(4)}  -- a3me constant (42),
roid-not-limited  RELATIVE-OID ::= {roid-energy not-limited(1)}  -- a3me constant (43),
roid-battery  RELATIVE-OID ::= {roid-energy battery(2)}  -- a3me constant (44),
roid-renewable  RELATIVE-OID ::= {roid-energy renewable(3)}  -- a3me constant (45),
roid-passive  RELATIVE-OID ::= {roid-energy passive(4)}  -- a3me constant (46),
roid-other-energy RELATIVE-OID ::= {roid-energy other-energy(0)}  -- a3me constant (47),
roid-storage  RELATIVE-OID ::= {roid-capability storage(5)}  -- a3me constant (48),
roid-rom  RELATIVE-OID ::= {roid-storage rom(1)}  -- a3me constant (49),
roid-ram  RELATIVE-OID ::= {roid-storage ram(2)}  -- a3me constant (50),
roid-flash  RELATIVE-OID ::= {roid-storage flash(3)}  -- a3me constant (51),
roid-hd  RELATIVE-OID ::= {roid-storage hd(4)}  -- a3me constant (52),
roid-other-storage  RELATIVE-OID ::= {roid-storage other-storage(0)}  -- a3me constant (53),
roid-communication  RELATIVE-OID ::= {roid-capability communication(6)}  -- a3me constant (54),
roid-computing  RELATIVE-OID ::= {roid-capability computing(7)}  -- a3me constant (55),
roid-other-capability RELATIVE-OID ::= {roid-capability other-capability(0)}  -- a3me constant (56),

roid-service  RELATIVE-OID ::= {roid-a3me service(3)}  -- a3me constant (57),
roid-hardware-service RELATIVE-OID ::= {roid-service hardware-service(1)}  -- a3me constant (58),
roid-software-service RELATIVE-OID ::= {roid-service software-service(2)}  -- a3me constant (59),
roid-real-world-service RELATIVE-OID ::= {roid-service real-world-service(3)}  -- a3me constant (60),
roid-other-service  RELATIVE-OID ::= {roid-service other-service(0)}  -- a3me constant (61),

roid-data-value RELATIVE-OID ::= {roid-a3me data-value(4)}  -- a3me constant (62),
roid-number RELATIVE-OID ::= {roid-data-value number(1)}  -- a3me constant (63),
roid-text RELATIVE-OID ::= {roid-data-value text(2)}  -- a3me constant (64),
roid-date RELATIVE-OID ::= {roid-data-value date(3)}  -- a3me constant (65),
roid-record RELATIVE-OID ::= {roid-data-value record(4)}  -- a3me constant (66),
roid-array  RELATIVE-OID ::= {roid-data-value array(5)}  -- a3me constant (67),
roid-stream RELATIVE-OID ::= {roid-data-value stream(6)}  -- a3me constant (68),
roid-other-data RELATIVE-OID ::= {roid-data-value other-data(0)}  -- a3me constant (69),

roid-property RELATIVE-OID ::= {roid-a3me property(5)} -- a3me constant (70),
roid-other  RELATIVE-OID ::= {roid-a3me other(0)}  -- a3me constant (71),
```

```
END
```

Listing 46: a3meOntologyOIDs.asn

A.6 A3ME Language Grammar in EBNF

```
/**
 * Author: Arthur Herzog
 * Created: Mon Dec 20 14:20:52 CET 2010
 */

Message-content ::= (
    Request-content
/* other message types are not handled by the A3ME-QL
  | Inform-content
  | Refuse-content
  | Not-understood-content
  | Cancel-content
  | Encrypted-content */
)

Request-content ::= ( "REQUEST"
    What
    (From-clause    )?
    (Condition-clause )?
    (Period-clause    )?
    (Range-clause   )?
)

Inform-content  ::= (
  ("sequence-number" Digits)?
  Resultset
)

Encrypted-content ::= (
  ("algorithm"    String)?
  ("encrypted-for"  String)?
  "ciphertext"  (Byte)+
)

What ::= (
    Data-descriptors
  | Service-call
)

Data-descriptors  ::= Data-descriptor (',' Data-descriptor)*

Data-descriptor   ::= A3ME-code '.' Infotype

Service-call ::= (
  "service-call" (
      "ID" Digits
    | "CAPABILITY" A3ME-code
  )
  "command" Digits
  ("parameters" Record)?
)

Infotype  ::= (
    "type-code"
  | "type-name"
  | "name"
  | "description"
  | "id"
  | "data"
  | "m2m-description"
  | "oid"
)
From-clause   ::= "FROM" DaID (',' DaID)*
```

```
DaID      ::= (
  "name" String
  ('(' Addresses ')')?
)

Addresses   ::= Address (',' Address)*

Address     ::= (
  "type"  String
  "address" String
)

Condition-clause  ::= "WHERE" Condition ("AND" Condition)*

Condition   ::= (
    "is-a"    A3ME-code
  | "is-for"  Is-for-condition
  | Operator-condition
)

Is-for-condition  ::= (
  Data-descriptor "FOR" A3ME-code
)

Operator-condition  ::= (
  Data-descriptor Operator Data-item
)

/*Is-a-condition  ::= (
  "IS-A" A3ME-code
)*/

Operator  ::= (
    "="   /*equals*/
  | ">"   /*greater*/
  | ">="  /*greater-equal*/
  | "<"   /*smaller*/
  | "<="  /*smaller-equal*/
)

Period-clause ::= (
  "period" Time-value
  ("duration" Time-value)?
)

Time-value  ::= (
  Digits Time-unit
)

Time-unit ::= (
    "ns"  /*nanosecond*/
  | "ms"  /*millisecond*/
  | "s"   /*second*/
  | "min"   /*minute*/
  | "h"   /*hour*/
  | "d"   /*day*/
  | "w"   /*week*/
  | "y"   /*year*/
)

Range-clause  ::= (
  "RANGE" Digits Distance-unit
)

Distance-unit ::= (
    "hop" /*hop*/
  | "m"   /*meter*/
  | "km"  /*kilometer*/
)
```

```
Resultset ::= (
  "schema"  Data-descriptors
  ("rows" Data-record (',' Data-record)* )?
)

Record    ::= '(' ( Data-item  (',' Data-item )* )? ')'

/* Record containing only data to be used inside resultset */
Data-record   ::= '(' ( Data (',' Data)* )? ')'

Data-item ::= (
  ("data-descriptor" Data-descriptor )?
  (Data)?
)

Data ::= (
    "integer-data" ('-')? Digits
  | "real-data" REALType
  | "boolean-data" ('t' | 'f')
  | "string-data" String
  | "date-data"   GeneralizedTimeString
  | "time-data" Time-data
  | "byte-data" Byte*
  | "bit-string" [0-1]*
  | "null"
  | "record-data" Record
)

REALType      ::= ( /* Definition from ASN.1-X.680 */
  ('-')? Digits '.' DIgits ('E' ('-')? Digits)?
)

Time-data ::= Time-value (',' Time-value)*

Refuse-content  ::= (
  (String)?
)

Not-understood-content  ::= (
  (String)?
)

Cancel-content  ::= (
  (String)?
)

String  ::= (Char | Digit)+

Char  ::= [a-z] | [A-Z]

/* YYYYMMDDHHMMSS.n[Z][-HHMM] */
GeneralizedTimeString ::= String

Byte  ::= [#x00-#xFF]
Digit ::= [0-9]
Digits  ::= Digits+
```

Listing 47: A3ME_QL.ebnf

A.7 Parser Definition for JavaCC to Translate A3ME-QL Queries into ASN.1 Notation

```
/**
 * JavaCC file for a3me-ql
 */
options
{
  JDK_VERSION = "1.5";
```

```
  static = false;
  OUTPUT_DIRECTORY = "a3me/ql";
  IGNORE_CASE = true;

  LOOKAHEAD= 4;
  //FORCE_LA_CHECK = true;
  //DEBUG_LOOKAHEAD= true ;
  //DEBUG_TOKEN_MANAGER=true;
}

PARSER_BEGIN(A3meQLParser)
package a3me.ql;
import java.io.StringReader;
import java.io.Reader;

public class A3meQLParser
{
  /**
  A String based constructor for ease of use.
  **/
  public A3meQLParser()
  {
  }

  public String parse(String s) throws ParseException{
    A3meQLParser parser = new A3meQLParser(new StringReader(s));
    parser.ReInit(new StringReader(s));
    return parser.expression();
  }

  public static void main(String args [])
  {
    try
    {
      String query = args [0];
      A3meQLParser parser = new A3meQLParser();
      parser.parse(query);
    }
    catch (Exception e)
    {
      e.printStackTrace();
    }
  }
}

PARSER_END(A3meQLParser)

SKIP :
{
  " "
| "\r"
| "\t"
| "\n"
}

TOKEN : /*RESERVED TOKENS FOR A3ME-QL */
{
  < REQUEST : "request" >
| < INFORM : "inform" >
| < REFUSE : "refuse" >
| < NOT_UNDERSTOOD : "not_understood" >
| < CANCEL : "cancel" >
| < ENCRYPTED : "encrypted" >
  //
| < FROM : "from" >
| < WHERE : "where" >
| < PERIOD : "period" >
| < DURATION : "duration" >
| < RANGE : "range" >
  //
```

```
  | < ALL : "all" >
  | < FOR : "for" >
  | < ISA : "isa" >
  | < AND : "and" >
  | < IDNR : "idnr" >
  | < SERVICECALL : "servicecall" >
  | < BYCAPABILITY : "bycapability" >
  | < COMMAND : "command" >
  | < PARAMETERS : "parameters" >
  }

  TOKEN : /* A3ME INFOTYPES */
  {
    < INFOTYPE :
      "type-code"
    | "type-name"
    | "name"
    | "description"
    | "id-nr"
    | "data"
    | "m2m-description" >
    //|    < DATA_DESCRIPTOR : <A3ME_CODE> <DOT> <INFOTYPE> >
  }

  TOKEN : /* A3ME CODEs */
  {
    //< A3ME_CODE : ["a"-"z"] (["a"-"z", "-", "3"])+ >
    < A3ME_CODE :
      "a3me"
    | "id"
    | "local-id"
    | "global-id"
    | "other-id"
    | "device"
    | "tag"
    | "mote"
    | "mobile"
    | "workstation"
    | "server"
    | "vehicle"
    | "multimedia"
    | "set-of-devices"
    | "other-device"
    | "capability"
    | "sensor"
    | "switch"
    | "temperature"
    | "light"
    | "humidity"
    | "acceleration"
    | "voltage"
    | "position"
    | "distance"
    | "sound"
    | "vision"
    | "vibration"
    | "radiation"
    | "chemical"
    | "other-sensor"
    | "actuator"
    | "switch-controller"
    | "device-controller"
    | "motion"
    | "manipulator"
    | "other-actuator"
    | "hid"
    | "input"
    | "output"
    | "inout"
    | "other-hid"
```

```
  | "energy"
  | "not-limited"
  | "battery"
  | "renewable"
  | "passive"
  | "other-energy"
  | "storage"
  | "rom"
  | "ram"
  | "flash"
  | "hd"
  | "other-storage"
  | "communication"
  | "computing"
  | "other-capability"
  | "service"
  | "hardware-service"
  | "software-service"
  | "real-world-service"
  | "other-service"
  | "data"
  | "number"
  | "text"
  | "date"
  | "record"
  | "array"
  | "stream"
  | "other-data"
  | "property"
  | "other" >
}

TOKEN : /* OPARATORS */
{
  < EQUALS : "=" >
| < GREATER : ">" >
| < GREATEREQUAL : ">=" >
| < SMALLER : "<" >
| < SMALLEREQUAL : "<=" >
}

TOKEN : /* Time-unit */
{
< TIMEUNIT :
  "nanosecond"
|   "millisecond"
|   "second"
|   "minute"
|   "hour"
|   "day"
|   "week"
|   "year"
>
}

TOKEN : /* Time-unit */
{
  < DISTANCEUNIT :
    "hop"
  | "meter"
  | "kilometer"
    >
}

TOKEN : /* SEPARATORS */
{
  < LPAREN : "(" >
| < RPAREN : ")" >
| < LBRACE : "{" >
| < RBRACE : "}" >
```

```
| < LBRACKET : "[" >
| < RBRACKET : "]" >
| < COLON : ":" >
| < SEMICOLON : ";" >
| < COMMA : "," >
| < DOT : "." >
}

TOKEN : /* data types */
{
  < STRING : ["A"-"Z"] ([ "A"-"Z", "0"-"9", "_", "-" ])+ >
| < QUOTED_STRING : "\"" (~[ "\"" ])+ "\"" >
| < CONSTANT : (< DIGIT >)+ >
| < #DIGIT : [ "0"-"9" ] >
}

String expression() :
{
  String str, content;
}
{
  {
    str = "asnmsg Message ::= {";
  }
  (
    content = content_request()
    {
      str += "\n  performative request, \n  content request-content : {";
    }
    //|   content_info()
    //|   content_refuse()
    //|   content_not_understood()
    //|   content_cancel()
  | content = content_encrypted()
    {
      str = "\n  performative inform, \n  content encrypted-content : {";
    }
  )
  [ < SEMICOLON > ]
  < EOF >
  {
    str += " " + content;
    str += "\n  }\n}";
    System.out.println(str);
    return str;
  }
}

String content_request() :
{
  String str, sWhat, sFrom = "", sWhere = "", sRepetition = "", sRange = "";
}
{
  < REQUEST > sWhat = what()
  [ sFrom = from() ]
  [ sWhere = where() ]
  [ sRepetition= repetition() ]
  [ sRange= range() ]
  {
    str = sWhat;
    str += sFrom;
    str += sWhere;
    str += sRepetition;
    str += sRange;
    return str;
  }
}

String what() :
{
```

```
  String str, tmp;
}
{
  tmp = data_columns()
  {
    str = "\n    what data-columns : {";
    str += tmp;
    str += "}";
    return str;
  }
| tmp = service_call()
  {
    str = "\n    what service-call : ";
    str += tmp;
    return str;
  }
}

String data_columns() :
{
  String str;
  String tmp1, tmp2;
}
{
  tmp1 = data_descriptor()
  {
    str = tmp1.toString();
  }
  (
    < COMMA > tmp2 = data_descriptor()
    {
      str += "," + tmp2;
    }
  )*
  {
    return str;
  }
}

String data_descriptor() :
{
  String str;
  Token t1, t2;
}
{
  t1 = < A3ME_CODE > < DOT > t2 = < INFOTYPE >
  {
    str = "{a3me-code " + t1 + ", infotype " + t2 + "}";
    return str;
  }
}

String service_call() :
{
  String str, sRecord = "";
  Token t1, t2, t3;
}
{
  < SERVICECALL >
  {
    str = "{\n      service";
  }
  (
    < IDNR > t1 = < CONSTANT >
    {
      str += " id : " + t1;
    }
  | < BYCAPABILITY > t1 = < A3ME_CODE >
    {
      str += " capability-code : " + t1;
```

```
    }
  )
  < COMMAND > t2 = < CONSTANT >
  {
    str += ", command " + t2;
  }
  (
  < PARAMETERS > t3 = < QUOTED_STRING > //sRecord = record()
  {
    str += ", parameters " + t3.image.replaceAll("\"", ""); //sRecord;
  }
  )?
  /*  service CHOICE {
    id INTEGER(0..65535),
    capability-code A3ME-code
    --service-code A3ME-code
  },
  command INTEGER(0..1023),
  parameters Record OPTIONAL
*/
  {
    str += "}";
    return str;
  }
}

String record() :
{
  String str, tmp1, tmp2;
}
{
  tmp1 = data_item()
  {
    str = tmp1.toString();
  }
  (
    < COMMA > tmp2 = data_item()
    {
      str += "," + tmp2;
    }
  )*
  {
    return "{" + str + "}";
  }
}


String data_item() :
{
  String str = "";
}
{

  {
    return str;
  }
}

String content_encrypted() :
{
  String str = "";
}
{
  < ENCRYPTED >
  // ...
  {
    return str;
  }
}
```

```
//[FROM (ALL / daID * [ "," daID ] ) ]
String from() :
{
  String str = "", sDaid;
  Token t1;
}
{
  < FROM >
  (
    str = daid()
    (
      < COMMA > sDaid = daid()
      {
        str += "," + sDaid;
      }
    )*
  | t1 = < ALL >
    {
      str = t1.toString();
    }
  )
  {
    if (str.equalsIgnoreCase("all")){
      return "";
    }
    else{
      str = ",\n    from {" + str + "}";
      return str;
    }
  }
}

String daid() :
{
  String str = "", tmp1, tmp2;
  Token t1;
}
{
  t1 = < STRING >
  {
    str = "{name \""+t1.toString()+"\"";
  }
  (
    < LPAREN >
    {
      str += " addresses {";
    }
    tmp1 = address()
    {
      str += tmp1;
    }
    (
      < COMMA > tmp2 = address()
      {
        str += "," + tmp2;
      }
    )*
    < RPAREN >
    {
      str += "}";
    }
  )?
  {
  str += "}";
    return str;
  }
}

String address() :
```

```
{
  String str = "";
  Token t1,t2=null,t3=null;
}
{
  t1=<STRING> <COLON>
  (
    t2=<STRING>
  | t3=<QUOTED_STRING>
  )
  {
  str = "{address-type \""+t1.toString()+"\"";
  if(t2 != null) str += ", address \""+t2.toString()+"\"";
  else if(t3 != null) str += ", address "+t3.toString();
  else str += ", address \"_\"";
  str += "}";
    return str;
  }
}

String where() :
{
  String str = "",sCondition;
}
{
  <WHERE>
  {
    str = ",\n    where {\n      ";
  }
  sCondition=condition()
  {
    str += sCondition;
  }
  (
    <AND>
    sCondition=condition()
    {
      str += ",\n      "+sCondition;
    }
  )*
  {
    str += "\n    }";
    return str;
  }
}

String condition() :
{
  String str = "";
  String sDD,sOp;
  Token code,tDI;
}
{
  (

  // for
  (sDD=data_descriptor() <FOR> code=<A3ME_CODE>
  {
    str = "is-for-condition {data-descriptor "+sDD+", a3me-code "+code+"}";
  }
  )
  // operator
  |   (
  sDD=data_descriptor()
  sOp=operator()
  (
    tDI=<QUOTED_STRING>
  | tDI=<CONSTANT>
  )
  {
```

```
    str = "operator-condition {operator "+sOp+", data-descriptor "+sDD+", parameter "+tDI+"}";
  }
  )
  // exist // note: must come as last otherwise data_descriptors are not recognized
  | (//LOOKAHEAD(<A3ME_CODE>, { getToken(2).kind != DOT } )
  <ISA> code=<A3ME_CODE>
  {
    str = "exists-condition : "+code;
  }
  )
  )
  {
    return str;
  }
}

String operator() :
{
  String str = "";
  Token t1;
}
{
  (

    t1=<EQUALS>
    {
      str = "equals";
    }
|   t1=<GREATER>
    {
      str = "greater";
    }
|   t1=<GREATEREQUAL>
    {
      str = "greater-equal";
    }
|   t1=<SMALLER>
    {
      str = "smaller";
    }
|   t1=<SMALLEREQUAL>
    {
      str = "smaller-equal";
    }
  )
  {
    return str;
  }
}

//repetition {period {number 1, time-unit minute}, duration {number 5, time-unit minute}},
String repetition() :
{
  String str = "";
  Token t1,t2,t3,t4;
}
{
  (
    < PERIOD > t1 = < CONSTANT > t2 = < TIMEUNIT >
    {
      str = ",\n    repetition {period {number " + t1 + ", time-unit " + t2 + "}";
    }
    (
    < DURATION > t3 = < CONSTANT > t4 = < TIMEUNIT >
    {
      str += ", duration {number " + t3 + ", time-unit " + t4 + "}";
    }
    )?
    {
      str += "}";
```

```
    }
  )
  {
    return str;
  }
}

//range {number 3, distance-unit hop}
String range() :
{
  String str = "";
  Token t1,t2;
}
{
  (
  < RANGE > t1=< CONSTANT > t2=< DISTANCEUNIT >
  {
    str = ",\n    range{number " +t1+", distance-unit "+t2+"}";
  }
  )
  {
    return str;
  }
}
```

Listing 48: a3meQLParser.jj